ALGEBRAIC AND NUMBER THEORETIC COMPUTING: ADVANCES AND APPLICATIONS IN VLSI SIGNAL PROCESSING

By

GLENN S. ZELNIKER

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1991

ACKNOWLEDGMENTS

First, I would like to thank my advisor and committee chair, Dr. Fred Taylor. Under his direction, I was given the freedom and autonomy to pursue whatever areas of research I found interesting. He also provided me with financial support, showed me the way in the academic world, and taught me the virtues of practicality.

I am indebted to Dr. R. E. Kalman for his two years of support at the Center for Mathematical System Theory. I would also like to thank my committee members, Dr. H. Lam, Dr. K. Sigmon, Dr. J. C. Principe, and Dr. D. Wilson. Special thanks go to Dr. Principe for many stimulating conversations about spectral estimation and adaptive filtering and to Dr. Wilson and Dr. Sigmon who instilled in me a longing to be a mathematician.

Monica Murphy at The Athena Group deserves special mention for financing much of the later work in this dissertation.

My girlfriend, Patricia, has made this past year one of incredible growth, both intellectual and personal. To her I am grateful for her companionship, support, and all of the wonderful things we have done together.

Finally, but most importantly, I must thank my family. They have given me constant support, love, and guidance. Without them, this work would not have been possible.

TABLE OF CONTENTS

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School of the University of Florida in Partial Fulfillment of the Requirements of the Degree of Doctor of Philosophy

ALGEBRAIC AND NUMBER THEORETIC COMPUTING: ADVANCES AND APPLICATIONS IN VLSI SIGNAL PROCESSING

By

Glenn S. Zelniker

May 1991

Chairman: Dr. Fred J. Taylor
Major Department: Electrical Engineering

Digital signal processing (DSP) is a field which has benefited from advances in device technology and integration. As devices are approaching their limits in terms of size and speed, the need for increased throughput still remains. It seems likely that the necessary breakthroughs will be at the arithmetic and algorithmic levels. We will show how the computational bottleneck can be freed by using innovations from the areas of algebraic and number theoretic computing. These innovations facilitate the development of fast algorithms and the design of new classes of arithmetic processors which are capable of achieving unprecedented throughput in a limited size with limited power dissipation.

The major sub-field of algebraic and number-theoretic computing we will discuss is the residue number system (RNS). The RNS is a system for computer arithmetic which is based on the principle of computation by homomorphic images (CHI). The RNS has been shown to be optimal in both size and speed and possesses some additional properties such as modularity and fault tolerance which make it an ideal candidate for VLSI signal processing implementations. We will also present some other CHI schemes for high-speed signal processing which are variants of the RNS and are easily implemented in VLSI.

This dissertation will provide the mathematical foundations of algebraic and number-theoretic computing and show how the RNS and its variants fit into the general theory. We also provide a new interpretation of the Chinese remainder theorem which yields new insight as to how it provides a decrease in computational complexity.

Finally, we will give general guidelines for the VLSI realization of RNS and other CHI hardware. As a concrete example, we will demonstrate the design of an RNS VLSI array processor for the computation of ultra high-speed fast Fourier transforms (FFTs). This processor serves as a motivation for the theory and points out the shortcomings of conventional technology. It will be seen that the processor is unmatched in terms of size, throughput, and power dissipation and illustrates the impact that algebraic and number-theoretic computing can have in designing extremely demanding systems for digital signal processing.

CHAPTER 1
INTRODUCTION

One of the principal advantages and applications of the modern digital computer is numeric data processing. High numeric data rate requirements are found in many technical areas such as signal and image processing, communications and radar, and artificial neural networks. The assumed performance metric in this field has been arithmetic speed measured in MIPs or MFLOPs. However, there are other factors which also must be considered in digital applications. First, many digital processors are designed to operate in a small volume and consume little power. Secondly, some of these digital arithmetic processors will operate unattended over long periods of time and must therefore carry with them the means of recovering from component failures.

All digital technologies bring with them their own agenda of speed, size, power, and fault-tolerance trade-offs. This dissertation will address some of the recent theoretical and technological advances which have led to the development of powerful new classes of numeric processors. One particular sub-area of algebraic computing will be treated in depth: the residue number system (RNS) and its many variants [67]. It will be shown that the RNS possesses a unique blend of speed, size, power consumption, fault tolerance, and arithmetic efficiency.

The use of the RNS will be motivated by the design of a machine for the ultra high-speed computation of fast Fourier Transforms, which is a problem of significant

interest to digital signal processing (DSP) and computer architecture researchers alike. We will explore all of the advantages and shortcomings of the RNS in an attempt to present a unified treatment of the subject and arrive at a clear understanding of how such a machine can be built. It will be shown that this machine, called the fast Fourier transform array processor (FFTAP), will be capable of achieving unprecedented performance and is vastly superior to processors predicated upon conventional technologies.

There have been many reports [67] on the design of RNS machines. The topics of these papers can be classified into the following fundamental problems of RNS design:

1. The design of binary-to-residue conversion units (input conversion).
2. The design of RNS computational engines for real and complex arithmetic.
3. The design of residue-to-binary conversion engines (output conversion).

Research reports in each of these areas have made many claims and recommendations regarding RNS component and system performance and design. In discussing the development of the FFTAP, we will address each of the above problems in an attempt to provide a set of guidelines which allow an RNS design which achieves an optimal balance of speed, size, and power.

One of the principal objections to the RNS has been the need for input and output conversion, which have traditionally been slow and hardware-hungry operations. Specifically, input data need to be converted from binary to residue format and output data needs to be converted from residue to binary format. This has presented a formidable problem for RNS researchers and has limited the use of the

RNS to application-specific devices (*i.e.*, devices which are hard-wired for a specific algorithm or application). We will respond to these objections and explore some recent developments which have greatly simplified data conversion. These advances should move the RNS into a position where it can not only interface with conventional systems, but provide a viable solution for a wide class of DSP applications.

The RNS, its many variants, and all of the other algebraic and number-theoretic computing techniques discussed herein are based on the concept of computation by homomorphic images (CHI) [46]. The basic principle of CHI is to embed computations in a computational structure in which they can be performed more simply than in their native domain. Conventional CHI schemes have relied on the language of rings, vector spaces, and finite fields, which are standard topics from Abstract Algebra [44]. We will also need to discuss some more sophisticated algebraic topics such as universal algebras [46] and associative algebras [33].

The work in this dissertation draws upon a wide body of research. The subject of algebraic computing can be traced to the work of algebraists and number theorists. Many of the algorithms and architectures we will use were inspired not only by research in the RNS, but by work pertaining to fast algorithms for digital signal processing [15], approximation theory [22], and error control coding [14]. The FFTAP, in addition to providing a motivation for the RNS, illustrates how a diverse body of technological and theoretical resources can be fused to solve a problem of great engineering importance.

1.1 History of Residue Number Systems

The genesis of the RNS is credited by Grosswald [30] and Knuth [41] to the Chinese general Sun Tzu, dating his work about 100 A.D. Although others date the

work considerably earlier, a book by Sun Tzu entitled *Suan-Ching* (Arithmetic Book) contained the following verse:

> We have things of which we do not know the number
> If we count them by threes, the remainder is two,
> If we count them by fives, the remainder is three,
> If we count them by sevens, the remainder is two,
> How many things are there?
> The answer, twenty-three.

How to obtain the answer 23 is outlined in Sun Tzu's book where a formula is suggested for manipulating remainders of an integer, after division by 3, 5, and 7. We acknowledge Sun Tzu's contribution by referring to the procedure for converting remainders into integers as the Chinese Remainder Theorem (CRT). Grosswald and Knuth also credit the Greek mathematician Nichomachus with independently discovering the CRT in about 100 A.D. During the Ming Dynasty (1368 A.D. - 1643 A.D.), Hsin Tai- Wei published a proof of the CRT as "Hun Hsin Tiang Bing" (Counting Soldiers) in *Suan Fa Tung Ching* (Unifying of All Counting Methods). This may have been the result of a belief that General Han Hsin used this method for counting troops. Hsin Tai-Wei's verse is as follows:

> Three men walk together, their chance of reaching seventy so slight.
> Among the five plum trees, twenty-one blossoms did they yield.
> Seven sons at midmonth, happily did reunite.
> Divide the sum by 105, the answer is revealed.

Cryptically (and somewhat unwittingly), Hsin Tai-Wei established an RNS system with a modulus set $\{3, 5, 7\}$ possessing a dynamic range of 105.

The study of the residue number system was made more mathematically rigorous when Euler presented a general proof of the CRT in 1734. This theorem, as well as the theory of modular arithmetic, was set forth in the 19th century by Carl Friedrich Gauss in his *Disquisitiones Arithmeticae* [25]. The theory of rings and fields developed by Cayley, Jordan, Hamilton, and Galois (to name but a few) made possible other breakthroughs in algebraic and number-theoretic computing. In fact, a great deal of this contemporary research is based on mathematical theory that predates the computer era by hundreds of years.

In the late 1950s and early 1960s, early work began on the RNS in general-purpose digital computing environments. Difficulties with sign detection, magnitude comparison, and division tended to offset the advantages of high-speed addition, subtraction, and multiplication in general purpose computing. The RNS did not prove to have clear advantages over conventional systems in general purpose computers so that researchers began to consider using the RNS in more restricted applications, such as digital signal processing.

By the late 1960s, digital signal processing was beginning to emerge as a technical subject distinct from general purpose computing. Digital hardware was becoming more integrated and low cost memories and ALUs became available on a single chip. The digital filter began to make its way out of the telephone industry and into avionics and consumer products. The arrival of the microprocessor in 1970 saw a flurry of activity in new DSP applications. It was at this time that investigators recognized that the RNS provides important advantages for signal processing where the computation is dominated by highly repetitive multiply/accumulate operations (*e.g.*, digital filtering, correlation, Fourier transforms).

The 1970s saw the advent of the VLSI era and the computer-on-a-chip brought many DSP techniques within practical reach. RAMs and ROMs grew in size and speed due to the VLSI technology. New problems arose, however, as integration grew larger. As memories grew denser and ICs grew more complex, it was sometimes quite difficult to test the internal functions of an integrated system from the limited terminals available to the external world. Designers were frustrated to find that many DSP algorithms which were developed for single instruction single data (SISD) machines did not partition in a way that allowed natural multi-chip implementation. It became apparent that new algorithms and design approaches were needed that would lead to modularity, parallelism, pipelined architectures, and fault tolerance. It was at this time that researchers began to re-examine old ideas from the RNS to see if they could solve some of the new problems brought on by VLSI technology.

In the case of the RNS, many of the variants on the basic RNS theory have been technology-based. That is, the RNS can now take advantage of smaller and faster digital computer chips. The theory of residue number systems is now finding its way into a number of arithmetic-intensive real-time applications such as radar, communications, adaptive beamforming, spectral analysis, and image processing. The RNS, in selected applications, has been shown to offer unparalleled integer (fixed-point) arithmetic speed, superior packaging, and reliability.

The speed advantage of the RNS is derived from the fact that arithmetic is performed in fast, carry-free small-wordlength channels operating in parallel. A packaging advantage can be gained by architecting a system as a set of compact short-wordlength devices rather than one large-wordlength processor. An RNS system can

also perform reliably in the presence of hardware failures since an error in one parallel channel can be isolated from all others.

Operations within the RNS can be classified into those that are fast and easy (addition, multiplication, subtraction), and those that are difficult (division, scaling, sign determination, magnitude comparison). Common DSP tasks such as filtering and spectral analysis are primarily characterized by large numbers of easy operations as well as the occasional scaling call. Since wordlengths are usually fixed and the dynamic range of the data is usually known, however, it is usually possible to pre-scale the calculations so that overflow does not occur. Also, image processing tasks such as edge detection and feature extraction require large numbers of simple repetitive operations that fall into the class of easy RNS operations.

In contrast, a general purpose programmable computer must handle a much broader set of computational tasks that require sign detection, magnitude comparison, and general division, operations which are difficult for the RNS. The flexibility required of a general purpose processor does not allow the overall system to take advantage of the inherent efficiencies of the RNS. For this reason, the RNS has not found any significant application in general purpose computing.

The usefulness of the RNS needs to be addressed in light of the fact that it has never seen widespread use in the design of general purpose computers. The inherent modularity of the RNS translates into a natural partitioning of the hardware that is particularly effective in VLSI designs. Since much of the current activity in DSP involves finding algorithms that are suitable for VLSI implementation, the RNS is an attractive approach. It is clear that in the field of DSP, where very fast computations are required, the RNS has a very important role to play. The modern requirements of

real-time DSP and the capabilities of VLSI implementation form an ideal setting for RNS techniques, which are capable of high computational speeds, modularity, fault tolerance, and simplified complex arithmetic.

Many of the other developments we will discuss are essentially extensions of the RNS. That is to say, they are usually based on the Chinese remainder theorem and are concerned with either modularizing algorithms, arithmetic, or both. This modularization will always result in parallelism, efficient hardware realization, and an increase in computational throughput.

1.2 Literature Survey

The first published work on residue number systems in the context of electronic computers was that of Svoboda and Valach in Czechoslovakia in the late 1950s [72, 70, 71]. They were interested in error control codes and studied them on a hard-wired small modulus computer. At roughly the same time, Garner [24], who was working in the United States, wrote about the basic concepts of residue arithmetic and showed how it could be useful as applied to electronic computers.

Other publications in the late 1950s and early 1960s included work at RCA [10], Lockheed [75], Westinghouse [62], and the Harvard Computation Laboratory [5]. The majority of this work tried to develop the theory of residue arithmetic and show how it could be useful for general purpose digital computers. Much of this work was supported by the U.S. Air Force and was not reported in the open literature because of Air Force restrictions on publication. The 1967 text by Szabo and Tanaka [73] (long since out of print and difficult to find) is a fairly accurate account of the developments that occurred during this early period of research.

The first use of the RNS in a digital signal processing environment was reported by Cheny [17] who designed a digital correlator based on residue arithmetic. Fairly little attention was given to the RNS in digital signal processing following Cheny's work because integrated circuit technology had not yet developed to the point where implementation was practical. Theoretical work, however, still continued, as exemplified by some of the papers on error detection and correction [81, 47, 88]

In the 1970s, digital signal processing was becoming an important discipline of its own and there was a large body of work on the RNS in digital signal processing systems. The majority of the early work in this is concentrated on input and output conversion and base extension [7, 52, 28].

The next wave of publication saw the design of RNS multipliers and use of RNS arithmetic in digital signal processing systems. The implementations varied, ranging from multiple microprocessors and read-only memory structures to custom VLSI solutions. Several papers attempted to take advantage of special moduli choices [78, 79], while others concentrated on ROM-intensive solutions that allowed arbitrary moduli choices [65].

A significant drawback of the RNS and its various extensions is the problem of residue-to-decimal conversion. This operation is necessary in order to recover results from the RNS as well as for scaling partial results (which is necessary since the RNS is a fixed-point system). Techniques such as the Chinese remainder theorem, mixed radix conversion, and core function techniques [67] are slow, expensive, and difficult to realize in hardware. Several recent developments rely on distributed arithmetic and on CRT approximation techniques [29]. These innovations have alleviated the

burden imposed by residue-to-decimal conversion and have made the RNS even more feasible.

Jenkins and Leon presented one of the first comprehensive treatments of RNS digital filters using modern integrated circuit techniques [38]. In 1978, Jenkins described a bit-slice RNS structure for realizing FIR filters [34]. This approach drastically reduced the memory requirements for RNS digital filters, but also reduced the throughput. The use of the RNS FIR in adaptive transversal filters has been studied by Soderstrand *et al.* [66] and Weinman *et al.* [82]. The latter paper used RNS ROM table look-up techniques to implement a transversal filter which is adapted using a modified LMS algorithm.

The use of the RNS in IIR filtering is considerably more complicated than in FIR filtering. Finite wordlength effects in an IIR filter are a serious problem in any fixed-point implementation, and scaling becomes an important issue. Some of the more important papers regarding the RNS in an IIR filtering environment include Soderstrand's work [63] on avoiding CRT scaling by performing scaling after every multiply by table look-up, and a paper by Jenkins [37] who proposed an RNS scaling algorithm which can be used quite effectively in the implementation of IIR filters.

Since digital signal processing is a complex arithmetic intensive field, work began toward developing efficient schemes for modular complex arithmetic. Early work in this area treated complex arithmetic in a manner similar to conventional complex arithmetic [76, 80]. That is, a complex multiply operation required four real multiplies and two real additions. Subsequent work attempted to embed computations in finite fields over which -1 is a quadratic non-residue [35, 8]. This work was important in that it led to a multiplier-free realization for complex arithmetic. Unfortunately,

these solutions were impractical because of the massive amounts of ROM needed to achieve any appreciable dynamic range.

The quadratic residue number system (QRNS) [42, 64] provided an elegant solution to the problem of reducing the amount of ROM needed for indexed complex arithmetic. The QRNS allowed multiplier-free complex arithmetic with only a small amount of ROM required and rendered residue computing a highly useful method for performing complex arithmetic intensive signal processing tasks.

A major advantage of the RNS is the size of the wordwidths in each RNS channel which translates directly into small arithmetic units. As a consequence, it is possible to fit multiple arithmetic units on a single chip. Since the RNS is best suited to long sums-of-products, a systolic architecture provides an attractive medium for RNS insertion. Systolic architectures have been shown [43] to have a significant speed/size advantage in applications where they can use a system of identical processing elements having regular dataflow, small wordwidth data paths, and where they can use sums-of-products or other simple and regular algorithms.

The parallelism at the arithmetic level provided by the RNS in addition to the parallelism at the architectural level provided by the systolic architecture can lead to a design with extremely high throughput, small size, and fault tolerance without introducing any additional hardware complexity. The use of the RNS in systolic architectures for filtering and transforms has been reported [39, 11, 16, 19, 45]. Jullien and his group at the University of Windsor have been involved in the design of bit-level systolic architectures for DSP [40, 84, 85, 74]. The author has been involved

in the development of an FFT array processor based on the QRNS [90] and a two-dimensional fault-tolerant VLSI systolic array for the computation of convolution, correlation, and other linear-algebraic operations [77].

Another important feature of the RNS is its ability to maintain a throughput which is independent of wordwidth and a gate complexity which does not grow drastically with wordwidth. Langston and Hinman [45] compared a conventional number system to the RNS for the design of a 256-tap systolic FIR. They used an eight-modulus system with five-bit moduli. Their analysis showed that, in general, the RNS hardware complexity grows approximately linearly with output wordwidth, whereas the complexity grows quadratically for conventional arithmetic systems. Additionally, the RNS was capable of maintaining a throughput which was independent of output wordwidth.

The use of the RNS in linear algebraic operations was considered by Young, Gregory, Krishnamurthy, and Howell [89, 27, 32]. Stallings and Bouillion [69] considered the computation of pseudoinverses using residue arithmetic and Alagar and Roy [6] were concerned with the computation of Smith normal form of integral matrices using residue arithmetic. The computation of matrix-matrix and matrix-vector products fall into the class of operations which are easily performed in the RNS. Other computations, such as eigenstructure, matrix inversion, and singular value decomposition are more difficult to perform in the RNS. Unfortunately, the most serious problem associated with integral matrix inversion, namely dynamic range expansion, has yet to be solved.

An extension of the QRNS, the polynomial residue number system (PRNS), was introduced by Skavantzos [60] in 1987. The purpose of the PRNS was to exploit the

Chinese remainder theorem [33] over polynomial rings to obtain a simplified realization for polynomial multiplication. This was a notion already familiar to those designing algorithms for high speed convolution, correlation, and Fourier transforms [50, 15, 46]. What is significant is that the PRNS provided efficient hardware realizations for parallel polynomial operations. Furthermore, it was shown that the PRNS meets Winograd's lower bound for multiplicative complexity [86].

Skavantzos and Stouraitis also showed how the PRNS could be used to design high-precision large-wordwidth real and complex multipliers. Similar in spirit is the work with the Algebraic Integer residue number system (AIRNS). Cozzens and Finkelstein showed how rings of algebraic integers can approximate complex numbers and be used to compute discrete Fourier transforms [20]. The complex-to-AIRNS mapping was treated in depth by Games [22, 21] and by a group at MITRE [23]. Although the AIRNS leads to an elegant implementation for complex arithmetic without the dynamic range problems associated with the RNS, a simply realizable complex-to-AIRNS mapping has yet to be developed.

The PRNS and Chinese remainder theorem were studied in depth by this author [94, 91, 92]. The PRNS was shown to be very useful in performing computations with polynomials over the complex field. Additionally, it was shown that the Chinese remainder theorem has an important interpretation in the context of associative algebras [93].

The use of number theory in digital signal processing can be traced to early work in computing fast Fourier transforms and convolutions. Researchers who were interested in spectral analysis or coding theory realized that conventional algorithms

were inefficient for their applications and that advances from algebra and number theory would be necessary.

During the 1970s a considerable amount of work was reported on number-theoretic transforms (NTTs). Since these NTTs are defined over a real or complex finite ring and attempt to reduce computational and hardware complexity, they appear to be related to RNS arithmetic. The relationship between the NTT and RNS arithmetic is analogous to the relationship between the FFT and conventional complex arithmetic. The NTT can be thought of as a special fast transform which has an FFT structure and a convolutional property, and which is typically defined within a single modulus RNS. The books by McClellan and Rader [50] and Blahut [15] give comprehensive treatments of NTTs.

The fast Fourier transform (FFT) over the complex field is usually credited to Cooley and Tukey [18], whose work began a flood of research in the area of fast transforms. It is known, however, that Good and Thomas had discovered an FFT algorithm based on the Chinese remainder theorem as early as 1960 [26]. Rader provided a method for discrete Fourier transform into a cyclic convolution [56]. Winograd [87] and Rader and Brenner [58] refined the ideas of Rader's prime algorithm to work with prime-power blocklengths.

Rader was the first to point out that a real convolution could be computed by embedding it in a field of integers modulo a Fermat or Mersenne prime [57]. Agarwal and Burrus also proposed the use of transforms based on Fermat primes [1, 2, 3] and implementation issues of these transforms were studied by McClellan [49]. The use of complex extensions of Galois fields to perform complex convolutions was first suggested by Nussbaumer [51] and by Reed and Truong [59]. Other number-theoretic

methods for convolution were proposed by Agarwal and Cooley [4]. For a comprehensive treatment, the reader is referred to Blahut [15] or Berlekamp [12].

1.3 Residue Number Systems in DSP

Since we have established that the RNS performs best in special purpose computationally intensive environments, we shall focus on the one that can take maximum advantage of the RNS's strengths, namely, DSP.

The four DSP operations that are of considerable interest to us are convolution, correlation, discrete Fourier transforms, and matrix products. For completeness, each of these operations is described below.

Convolution: If $\{h_k\}$ is an N-point sequence and $\{x_k\}$ is an L-point sequence, the linear convolution of h and x is the $L+N-2$-point sequence $\{y_k\}$ where

$$y_k = \sum_{i=0}^{N-1} h_i x_{k-i}.$$

Correlation: If $\{x_k\}$ and $\{y_k\}$ are N-point sequences, the correlation of x and y is the $2N-2$-point sequence r where

$$r_k = \sum_{i=0}^{N-k-1} x_k y_{n+k}.$$

DFT: If $\{x_k\}$ is an N-point sequence, the DFT of x is the N-point sequence, X, where

$$X_n = \sum_{k=0}^{N-1} x_k e^{-j2\pi nk/N}.$$

Matrix Product: If A is an $m \times l$ matrix and B is an $l \times n$ matrix, then the matrix product of A and B is the $m \times n$ matrix C where

$$c_{ij} = \sum_{k=1}^{l} a_{ik} b_{kj}.$$

All of these operations have a common bilinear structure [86].

Traditional means of implementing the above operations in real time involved either developing an efficient algorithm or a fast system architecture. Direct computation, however, may not be the most efficient. Historically, multiplication was the slowest operation to implement in hardware. Hence, algorithms were designed to minimize multiplication count, which is still a measure of computational bandwidth. On the other hand, algorithm efficiency usually means a more complex algorithm, *e.g.*, the decimation in time FFT. Here, the computational complexity is drastically reduced, but the data handling is far more complex than direct computation of the DFT.

In general, algorithm efficiency tends to increase hardware complexity, which in turn reduces operating speed. Therefore, real-time implementation of efficient DSP algorithms has relied on exotic and expensive machines which are dedicated to a single task. DSP chips have, to a degree, eased this problem, but are still relatively slow for most real-time applications. Even with the availability of special-purpose processors, the requirements of some DSP operations are still too severe to be met with current technology.

The design of a high performance DSP machine must therefore meet two conflicting objectives: algorithm efficiency and simple high speed architecture. A balance must be met which utilizes the best attributes of both algorithms and architectures. This is difficult to achieve with conventional technology. The RNS, however, eases the conflict by increasing computational bandwidth drastically.

It is essential to establish a set of primitive RNS operations with which to determine the optimal algorithm/architecture synergism. As discussed earlier, the RNS is

ideally suited to performing addition, subtraction, and multiplication. Difficult operations include magnitude comparison, general division, and overflow detection. The difficult operations usually require that an algorithm jump between RNS and integer representations. We shall confine our attention to algorithms which use a majority of easy RNS operations. Thus, we define the RNS primitive operations to be addition, subtraction, and multiplication. Although this may seem somewhat restrictive, it is sufficient for all of the fundamental DSP operations discussed previously. The RNS primitives fit naturally into a sum-of-product structure, which has a simple and efficient hardware realization. Therefore, we can identify a structural primitive as the sum-of-product.

As a final note, there is a difference between arithmetic advantage and hardware efficiency. For example, an isolated multiply-accumulate operation in the RNS would require an input converter, a computational engine, and an output converter. It is clearly impractical to replace a conventional multiply-accumulator with an RNS unit in this situation. If there are many sequential RNS operations, however, the amount of hardware and latency due to the conversion becomes less significant. Therefore, to benefit from the RNS, we shall assume that the amount of time spent performing RNS operations is much greater than the time spent converting to and from the RNS. Again, this is quite true for all of the fundamental DSP operations we have considered thusfar.

In subsequent chapters, we will develop the design of an RNS multiply/accumulate (MAC) engine, which is the heart of any DSP processor. The MAC engine is the building block of the FIR, which is the basic structure for implementation of all of our fundamental DSP operations. We will use a direct implementation of the FIR for

two reasons. First, it is easy to vary the FIR length by cascading FIR substructures. Second, the direct FIR implementation maps easily to systolic architectures. While this does not take advantage of a reduced multiplication count, it is important to realize that we can perform multiplication at the same speed as addition (this will be shown).

1.4 The Need for an Alternative Technology

The computation of discrete Fourier transforms (DFTs) at high bandwidths is a problem of great importance in engineering areas such as time-varying signal analysis, vibration analysis, spectral estimation, and Doppler signal processing. New algorithms in these areas often require the production of DFTs at rates far in excess of those obtainable given present technological constraints.

Furthermore, there is a current trend toward "sensor fusion." In many cases, it is desirable to place some of the computational burden of complex signal processing systems on the sensors themselves. That is to say, the sensors are integrated with processors that are capable of performing front-end signal processing tasks (*e.g.*, high-speed digital filtering, transform computation) autonomously. These front-end processors need to be compact, dissipate little power, run at high speeds, and be relatively fault tolerant.

The mid- to late-1980s saw the advent of the digital signal processing (DSP) microprocessor as a viable tool for demanding signal processing problems. In the years that followed, the DSP microprocessor evolved from a slow, primitive, fixed-point processor to a sophisticated, high-speed, floating point device capable of performing demanding and high-precision signal processing tasks in a compact package. The

evolution of the DSP microprocessor is well illustrated by the Texas Instruments TMS320 series.

The first generation of DSP microprocessors was fixed-point in nature and carried on-chip data acquisition (*i.e.*, ADC/DAC). Second generation devices replaced the on-chip data acquisition subsystems with expanded memory space and were exemplified by the Texas Instruments TMS3201X series of processors. The devices had instruction cycle times on the order of 200 ns and were capable of performing roughly 5 to 15 million instructions per second (MIPS). It was possible to compute 1024- point FFTs in roughly 10 ms, which is far below the needs of most real-time signal processing systems.

The current generation of DSP microprocessors is characterized by several characteristics due to improvements in process technology (these devices often have close to one million transistors on a chip) 32-bit parallel floating-point multipliers and accumulators capable of performing a multiply-accumulate operation in one clock cycle. The inclusion of fast, compact, on-chip memory enabled the high-speed execution of programs, and, coupled with on-chip direct memory access (DMA) controllers, data could be moved quickly and easily to and from the processor.

These processors also feature low-overhead hardware looping, which, when coupled with fast floating-point arithmetic, allow the high-speed execution of iterative algorithms without the problems associated with fixed-point processors due to finite wordlength effects. Additionally, these DSP microprocessors were endowed with large instruction sets which allowed the engineer to easily program FIRs, IIRs, and fast Fourier transforms (FFTs). Many of these newer DSP microprocessors ran in the 30 MHz range and were capable of roughly 25 MIPS. A good example of this class of

processor is the TI TMS320C30 series. A reasonable benchmark for the production of a 1024-point using this class of processor is roughly 3 ms. This processor, although much faster and more accurate than earlier designs, still falls far short of the needs of real-time signal processing systems.

Recently, there has been a return to fixed-point designs in an attempt to gain a wide speed advantage over the more complex floating-point processors just discussed. AT&T's DSP 16A and Texas Instruments' TMS320C50 are both high precision fixed-point processors capable of achieving nearly 40 MIPS. Although this brings the computation of a 1024-point FFT to a sub-millisecond procedure, this is still far short of the rates required by most high-end signal processing applications.

One problem with conventional DSP microprocessor design is the stagnation of multiplier technology. Most DSP operations are multiply-accumulate intensive and hence the processor's computational power is largely dependent upon the speed of its multiply-accumulator (MAC). What is more, many DSP algorithms demand the use of complex arithmetic, which requires several MAC cycles per complex operation.

Typically, up to 40 percent of the silicon area of a DSP microprocessor is dedicated to the MAC. Arithmetic throughput could be bolstered by the inclusion of more MACs, but this is frequently impossible due to area and power constraints. Both inMOS and Array Microsystems manufacture processors which feature multiple MACs but are low-precision, small-wordwidth designs. Even with the inclusion of MAC arrays, the FFT cycle times of both devices still fall drastically short of what is needed for highly demanding applications.

The only commercial FFT processor that comes close to meeting the requirements of a high-end, real-time signal processing system is the FFT chipset from Plessey

which is capable of computing a 1024-point FFT in roughly 96 μs. This translates into approximately 10K 1024-point transforms per second. We will show, however, that with only a moderate hardware investment we can do much better than this.

Another approach to increasing arithmetic throughput is through the use of exotic technologies such as emitter coupled logic (ECL) or Gallium Arsenide (GaAs). Although ECL and GaAs memories and logic parts can be run at speeds approaching 1 ns, the power dissipation of these technologies is enormous. For example, Gigabit Logic manufactures the 14GM048, a 512 by 8-bit ROM which has an access time of 1 ns but consumes nearly 3W of power. Also, neither GaAs nor ECL can be packaged densely enough to build practical signal processing systems. Designs built from discrete components would be too large and dissipate too much power to be a front-end signal processing reality.

There are also obstacles to high-throughput signal processing which are due to algorithms and architectures. Single-instruction single-data (SISD) architectures have a throughput which is inherently limited by computational and input-output bottlenecks. It seems evident that in order to achieve real-time bandwidths, it will be necessary to move to single-instruction multiple-data (SIMD) or multiple-instruction multiple-data (MIMD) architectures. The problem with using conventional DSP microprocessors as computational elements in such architectures is the complexity of design and reliability. DSP chips are typically packaged in 100 or more pin packages and feature a data-flow which is too irregular to integrate into an array.

If an array were to be constructed from conventional arithmetic units or DSP microprocessor chips, the clock rate of the array would still be limited by the throughput of these computational elements, which is typically limited by the MAC speed. Even

if the design is highly pipelined, the machine can not run any faster than the throughput rate determined by the MAC throughput.

It appears, then, that the development of a high-speed, compact, and low power dissipating FFT processor should be based on a SIMD or MIMD architecture of small, reliable, high-speed, high-throughput computational elements that can be manufactured in a low power technology such as CMOS. Moreover, these computational elements should feature a high degree of multiply/accumulate efficiency and be capable of supporting complex arithmetic with little or no overhead. We will discuss the design of an FFT array processor which possesses all of the above attributes.

We call the machine the fast Fourier transform array processor (FFTAP). The FFTAP is based on the quadratic residue number system (QRNS), a variant of the RNS which will be discussed in the following chapters. The QRNS is based on modular operations performed concurrently in many parallel small-wordwidth channels, is table look-up intensive, and leads to a realization for complex arithmetic primitives which is multiplier-free. Essentially, a multiplier is replaced by a binary adder. This is important since it allows us to place many multipliers in a single VLSI package, providing a significant advantage in functionality over conventional arithmetic systems. Also, the arithmetic within QRNS units is highly pipelined which leads to extremely high throughput and makes the QRNS an ideal candidate for insertion into array architectures.

The QRNS is capable of maintaining a hardware complexity which is essentially a linear function of wordwidth. Even more desirable, the throughput is independent of the wordwidth (numerical precision of the operands). Furthermore, the QRNS provides a hardware realization for finite-field ALUs which is highly repetitive and

modular. The QRNS is based on many tiny and identical computational units working in parallel. The computational units consist of nothing more than small tables and adders and can be pipelined at the rate of the tables, which are typically slower. The data flow between these computational units is highly regular and amenable to VLSI implementation.

It will be shown that the QRNS is to be an ideal number system complex arithmetic-intensive operations such as those found in the FFT. The QRNS reduces the operation count of complex multiplication by eliminating all coupling between real and imaginary channels. This creates a reduction in interconnection within the computational elements, which is important in VLSI design. Using the QRNS, we can build complex multiply/accumulate engines which operate as much as ten times faster than the conventional state-of-the-art while using as little as one-tenth of the area. These ALUs are highly pipelineable and have the additional property of graceful degradation. We will see that the QRNS makes possible the VLSI implementation of a high-speed FFT array processor which has unprecedented throughput, size, and power dissipation.

Since the RNS (and the QRNS) is a fixed-point system, there have been significant problems with loss of precision. Because residue number systems are not just unweighted, but modular as well, there is no acceptable manner in which to handle dynamic range overflow. For each nested multiply operation, there is a geometric growth in the dynamic range. Countering this problem requires conversion from the RNS to the integer system, scaling, and returning to the RNS for further processing.

Scaling has been one of the principal limitations of RNS system design. The required RNS to integer conversion is slow and hardware-intensive and relies on the design of large custom modulo-M adders. The efficacy of the RNS is limited by the

number of scaling calls to manage the dynamic range and hence the RNS's utility was typically limited to long sums-of-products. Because of this, most RNS and QRNS DFT processors depended on the use of convolution form DFTs such as the Goertzel algorithm or the Rader prime algorithm. While this did result in a realizable design, it did not take advantage of the computational efficiency provided by decimation-in-time algorithms.

Decimation-in-time FFT algorithms, however, are based on nested multiplies and hence suffer from dynamic range problems. To take advantage of the drastic reduction in multiplicative complexity afforded by such algorithms, we have developed an RNS scaling algorithm which uses only standard binary hardware and can be pipelined at the RNS's computational speed. This moves the RNS and QRNS into a middle ground where they are capable of extremely high throughput while executing a broader class of algorithms than was previously possible.

1.5 Organization of Dissertation

Chapter 2 gives the formal mathematical framework for the remainder of this dissertation. The chapter begins with the introduction of universal algebras and explores the underlying principle of most of what follows: computation by homomorphic images (CHI). It will be seen that CHI is a very powerful principle which facilitates many advances in high-speed computing. The majority of our results in CHI is based on the Chinese remainder theorem (CRT), a classic result from the theory of rings. The CRT, in its most general form, will be used repeatedly to generate highly efficient algorithms, architectures, and arithmetic.

Chapter 3 discusses the residue number system (RNS). The RNS exploits the CRT at the arithmetic level and results in a parallel system for high-speed arithmetic. The

fundamental concepts of the RNS are described as well as efficient hardware realizations for RNS primitives. We also examine residue-to-decimal conversion, which is perhaps the most difficult of all RNS operations. Residue-to-decimal conversion has been one of the major factors which has limited the applicability of RNS solutions to a narrow area of numeric processing. New residue-to-decimal conversion techniques will be explored which are extremely fast, simple, and easy to implement. It is hoped that these innovations will move the RNS into a less specialized position in numeric processing.

Chapter 4 introduces the quadratic RNS (QRNS). The QRNS is a variant of the RNS which is optimized for complex arithmetic. The QRNS has found much use recently due to the complex arithmetic-intensive nature of digital signal processing (DSP). In this chapter, we present one of the more significant developments in recent RNS research: the multiplier-less QRNS. This innovation eliminates the need for general-purpose multipliers, which are typical very slow and extremely large. The elimination of multipliers from DSP hardware design allows the introduction of massively parallel architectures which are capable of unprecedented throughput in a minimal size.

The use of the RNS in digital filtering is studied in Chapter 5. Digital filtering is type of numeric processing which is dominated by long strings of multiply/accumulate operations, which are ideally suited to RNS implementation. Digital filtering is becoming such an important and widespread topic that it alone could justify the study of the RNS. We will present some extremely high-throughput systolic architectures for RNS digital filtering in this chapter.

Another recent development in algebraic computing is the polynomial RNS (PRNS). The PRNS exploits the CRT over polynomial rings to achieve a parallel decomposition at the *algorithmic* level rather than at the arithmetic level. The advantages of the PRNS can be paired with the advantages of the RNS or QRNS to generate highly efficient schemes for correlation, convolution, and discrete Fourier transforms (DFTs). We will present some new algorithms and architectures for the computation of one- and two-dimensional DFTs using both the PRNS and the QRNS. in Chapter 6.

The next chapter studies the issue of computational complexity as related to the CRT. It is known that the PRNS is capable of reducing the complexity of polynomial multiplication (and hence convolution, correlation, and DFTs) from $O(N^2)$ to $O(N)$. The QRNS reduces the computation of a complex multiply from four real multiplies and two real adds to just two real multiplies (or just two real adds in the case of the logarithmic QRNS). Traditional explanations for this reduction in complexity have centered on ring-theoretic arguments and do not make the reasons transparent. We present a study of multiplicative in the context of associative algebras which sheds new light on the subject. It will be quite clear why PRNS and QRNS meet Winograd's lower bound on complexity.

Chapter 8 discusses the algebraic integer RNS (AIRNS), which is another variant of the RNS. The AIRNS has been proposed as a solution to the dynamic range problems experienced by the RNS and the QRNS. The AIRNS is based on rings of algebraic integers based on cyclotomic extension fields. The principal justification for the use of the AIRNS is that certain rings of algebraic integers are dense in the complex plane, hence yielding arbitrarily accurate approximation for complex

numbers. We will show that the mappings associated with the AIRNS are quite difficult to realize in hardware. While the theory is quite elegant, the primary reason for the AIRNS is to eliminate scaling in the RNS. The introduction of new residue-to-decimal and scaling techniques, however, appear to alleviate the problem and render the AIRNS an ineffective alternative due to its hardware-intensive nature.

Chapters 9 and 10 explore the computation of fast Fourier transforms (FFTs). Chapter 10 discusses the use of a distributed arithmetic (non-RNS) technique for computing FFTs. It will be shown that an extremely fast compact FFT engine can be designed using distributed arithmetic. This engine, called the distributed arithmetic FFT (DAFFT) is based on the use of convolution-form FFTs, which do not take advantage of the drastic reduction in multiplicative complexity allowed by decimation-in-time or frequency algorithms, however.

In Chapter 11, we present the design of an FFT array processor based on the logarithmic QRNS which uses a radix-8 decimation-in-time FFT algorithm. This processor, called the FFT array processor (FFTAP), utilizes all of the advantages provided by the RNS as well as the new scaling and conversion techniques to achieve unmatched throughput in a single-board processor design. The FFTAP is a proof-of-concept that the RNS need not be limited to multiply/accumulate algorithms and has genuine utility in a more demanding computing environment. The FFTAP illustrates the practical uses of most of the techniques discussed in the dissertation and shows that the RNS can move into a middle ground where it is capable of serving a wider variety of computing needs.

Finally, we close with a summary of important results which were presented throughout the dissertation. This chapter attempts to unify all of our theoretical

and technology-related results into a clear picture which illustrates the impact and importance of algebraic and number-theoretic computing. In particular, we provide a synopsis of the most salient points and give general guidelines for the development and design of RNS-type machines. We believe that these guidelines lead to extremely simple and testable designs which provide low-cost, reliable, high-performance realizations of a technology which was previously thought to be "too exotic" for mainstream usage.

CHAPTER 2
MATHEMATICAL PRELIMINARIES

This section is a self-contained review of the mathematics necessary for an understanding of the material that follows. The majority of the material comes from the theory of commutative rings and finite fields. The only exception is the sections concerning associative algebras and universal algebras.

2.1 Universal Algebras

This material is fundamental to the theory of algebraic computing. Universal algebras and morphisms serve as the language with which to discuss algebraic computing and all of our results will be seen to be special cases of the general results contained in this section.

2.1.1 Algebraic Systems

As of this writing, there has not been a rigorous justification for the use of algebraic and number-theoretic computing as applied to digital signal processing. Rather, there is a tendency toward naive application of number-theoretic and algebraic "tricks" without a sound theoretical basis.

We choose here to discuss our work in an abstract setting, which although may seem unnecessary at first, will provide a sound theoretical footing for subsequent results. Fundamental concepts such as computations and expressions will first be defined in the context of universal algebras and a justification for computation via

homomorphic images will be clear. Much of this material can be found in references such as those by Lipson [46] or Birkhoff and Bartee [13].

To begin, we provide a definition of just what is meant by a universal or Ω-algebra.

Definition 2.1. *An Ω-algebra is a pair $[A, \Omega]$ where*

1. *A is a set.*
2. *Ω is a collection of operations defined on A. Each operation $\omega \in \Omega$ is a function from A^n to A taking operands $a_1, a_2, \ldots, a_n$ into $\omega(a_1, a_2, \ldots, a_n)$ where the arity, n of ω is a nonnegative integer that depends on ω.*

An algebraic structure that will be of considerable interest to us is the finite ring. It is assumed that the reader is familiar with the theory of rings, which is standard material from Abstract Algebra [44, 33]. A ring can be viewed as an Ω-algebra if the set R is identified with the set A and the collection of operations Ω is the set $\{+, \cdot\}$. In each of the subsequent, the relation to the more specific case of finite rings should be kept in mind.

Definition 2.2. *Let A be an Ω-algebra. A subset B of A is called an Ω-subalgebra if B is Ω-closed. That is, for any $\omega \in \Omega$,*

$$x_1, x_2, \ldots, x_n \in B \Longrightarrow \omega(x_1, x_2, \ldots, x_n) \in B.$$

This relationship is denoted by $B \leq A$.

Since we are interested in computing via homomorphic images, it is necessary to provide a way to relate different algebras. To achieve this, we need the notions of similar algebras and morphisms.

Definition 2.3. Two algebras $[A,\Omega]$ and $[A',\Omega']$ are called similar if there is a bijection from Ω to Ω' such that corresponding operations $\omega \in \Omega$, $\omega' \in \Omega'$ have the same arity.

Definition 2.4. Let $[A,\Omega]$ and $[A',\Omega']$ be similar algebras. A function $\phi : A \to A'$ is said to be a morphism from $[A,\Omega]$ to $[A',\Omega']$ if for any $\omega \in \Omega$ and for any $a_1, a_2, \ldots, a_n \in A$,

$$\phi(\omega(a_1, a_2, \ldots, a_n)) = \omega'(\phi(a_1), \phi(a_2), \ldots, \phi(a_n)).$$

Morphisms have the important property that they preserve subalgebras in the forward and inverse sense.

Theorem 2.1. Let A and A' be Ω-algebras and $\phi : A \to A'$ a morphism. Then

1. $B \leq A \Longrightarrow \phi(B) \leq \phi(A)$.
2. $C \leq A' \Longrightarrow \phi^{-1}(C) \leq A$.

As in more familiar structures such as sets or rings, equivalence and congruence relations are important in universal algebras.

Definition 2.5. A congruence relation E on an Ω-algebra A is an equivalence relation on A that satisfies the following. For any $\omega \in \Omega$,

$$a_i E b_i \quad \text{for} \quad i = 1, 2, \ldots, n \Longrightarrow \omega(a_1, a_2, \ldots, a_n) E \omega(b_1, b_2, \ldots, b_n).$$

We define the equivalence class of an element $a \in A$ to be the set of all $x \in A$ such that xEa. This is denoted by the symbol $[a]$.

We now show that quotient sets can be given the structure of universal algebras. This is fairly important in constructing schemes for homomorphic computing.

Theorem 2.2. Let A be an Ω-algebra and E a congruence relation on A. Then

1. *The quotient set A/E is an Ω-algebra if we define $\omega \in \Omega$ on A/E by*

$$\omega([a_1],[a_2],\ldots,[a_n]) = [\omega(a_1,a_2,\ldots,a_n)].$$

2. *A/E is a morphic image of A under the canonical map $\nu : a \mapsto [a]$.*

Lemma 2.1. Let $\phi : A \to A'$ be a morphism of Ω-algebras. Then the kernel relation E_ϕ,

$$aE_\phi b \iff \phi(a) = \phi(b)$$

is a congruence relation on A.

The following can be regarded as the fundamental theorem of morphisms of universal algebras.

Theorem 2.3. Let A and A' be Ω-algebras with A' a morphic image of A under ϕ. Then

$$A' \cong A/E_\phi$$

where $\psi : [a] \mapsto \phi(a)$ is an isomorphism of A/E_ϕ and A'.

2.1.2 Computation by Homomorphic Images

We must identify precisely what is meant by an expression. Naively, an expression is a series of computations to be performed in some algebraic structure. It will be clear that the our fundamental DSP operations can be regarded as expressions.

Definition 2.6. Let Ω be a set of operation symbols. Let $X_s = \{x_1, x_2, \ldots, x_s\}$ be a set of s distinct symbols. The set $\mathcal{E}(X_s)$ of all Ω-expressions $e(x_1, x_2, \ldots, x_s) = e(\mathbf{x})$ in the s indeterminates of X_s is defined recursively by

1. $x_1, x_2, \ldots, x_s \in \mathcal{E}(X_s)$

2. *For any* $\omega \in \Omega$,

$$e_1(\mathbf{x}), e_2(\mathbf{x}), \ldots, e_n(\mathbf{x}) \in \mathcal{E}(X_s) \Longrightarrow \omega(e_1(\mathbf{x}), e_2(\mathbf{x}), \ldots, e_n(\mathbf{x})) \in \mathcal{E}(X_s).$$

Definition 2.7. Let $e(x_1, x_2, \ldots, x_s) = e(\mathbf{x})$ *be an* Ω*-expression, let* A *be an* Ω*-algebra, and let* $a_1, a_2, \ldots, a_s \in A$. *Then the value* $e(a_1, a_2, \ldots, a_s) = e(\mathbf{a})$ *of* $e(\mathbf{x})$ *for* $x_i = a_i$ *is defined recursively by*

1. $e(\mathbf{x}) = x_i \Longrightarrow e(\mathbf{a}) = a_i$.

2. $e(\mathbf{x}) = \omega(e_1(\mathbf{x}), e_2(\mathbf{x}), \ldots, e_n(\mathbf{x})) \Longrightarrow e(\mathbf{a}) = \omega(e_1(\mathbf{a}), e_2(\mathbf{a}), \ldots, e_n(\mathbf{a}))$.

We arrive finally at the result that allows us to compute via homomorphic images. Roughly speaking, we have an expression that needs to be computed over an algebra A. There exists an algebra A' and a morphism ϕ between A and A'. The algebra A' will have the desirable property that the expression is much simpler to evaluate over A'. Thus, it is desired to map the indeterminates and the expression to A', evaluate the expression, and recover the result in A from that obtained in A'.

Theorem 2.4. Let $\phi : A \to A'$ *be a morphism of* Ω*-algebras and let* $e(x_1, x_2, \ldots, x_s) = e(\mathbf{x})$ *be an* Ω*-expression. Then*

$$\phi(e(\mathbf{a})) = e(\phi(\mathbf{a})).$$

2.2 The Chinese Remainder Theorem

The algebraic setting for much of our work is in commutative rings. Recall that a ring is a set, R with two binary operations, $+$ and $\cdot$. There is an identity 0 for R

under $+$ and an identity 1 for R under $\cdot$. R forms a group with the operation $+$, R is both commutative and associative under both $+$ and $\cdot$ and the operations distribute over one another.

An ideal is a subset J of R that is both a subgroup of R under $+$ and has the property that $RJ \subseteq J$. By the *ideal sum* of two ideals I_1 and I_2 of a ring R, is meant the set

$$I_1 + I_2 := \{i_1 + i_2 | i_1 \in I_1, i_2 \in I_2\} \subseteq R.$$

Theorem 2.5. Let R be a ring and let $I_1, I_2, \ldots, I_n \in R$ be a set of pairwise relatively prime ideals, i.e.,

$$I_k + \bigcap_{j \neq k} I_j = R. \tag{2.1}$$

If

$$I = \bigcap_{i=1}^{n} I_i,$$

then

$$R/I \cong R/I_1 \oplus R/I_2 \oplus \cdots \oplus R/I_n.$$

Proof: Let φ be the canonical homomorphism $\varphi : R \to R/I_1 \oplus \cdots \oplus R/I_n$ given by

$$\varphi(x) = (x \bmod I_1, \ldots, x \bmod I_n). \tag{2.2}$$

By assumption, the ideals $I_1, \ldots, I_n$ are pairwise relatively prime. Hence, for each i, there exist $\alpha_i \in I_i$ and $\beta_i \in \bigcap_{j \neq i} I_j$ such that

$$\alpha_i + \beta_i = 1. \tag{2.3}$$

This means that

$$\beta_i = 1 \bmod I_i \quad \text{and} \quad \beta_i = 0 \bmod I_j, \ j \neq i. \tag{2.4}$$

Let

$$(x_1, x_2, \ldots, x_n) \in R/I_1 \oplus \cdots \oplus R/I_n$$

and consider

$$x := \sum_{i=1}^{n} x_i \beta_i. \tag{2.5}$$

Then $\varphi(x) = (x_1, x_2, \ldots, x_n)$, so φ is surjective. Now, let $y \in \bigcap_{i=1}^{n} I_i$. Then $\varphi(y) = (0, 0, \ldots, 0)$. Also, $\varphi(y) = 0$ means that $y \bmod I_i = 0$ for all i. Thus,

$$ker(\varphi) = \bigcap_{i=1}^{n} I_i := I$$

and by the fundamental theorem of homomorphisms of rings,

$$R/I \cong R/I_1 \oplus \cdots \oplus R/I_n. \quad \square$$

2.2.1 The Integer CRT

Let R be the ring of integers, Z, and let the ideals I_i be the principal ideals (p_i) generated by the pairwise relatively prime integers $p_1, p_2, \ldots, p_L$ (called the *moduli*). The ideals satisfy Eq. 2.1 and their intersection is given by

$$\bigcap_{i=1}^{L} (p_i) = (M), \quad \text{where} \quad M = \prod_{i=1}^{L} p_i.$$

Then the Chinese Remainder Theorem says that

$$Z_M \cong Z_{p_1} \oplus Z_{p_2} \oplus \cdots \oplus Z_{p_L}.$$

The forward mapping is given by Eq. 2.2,

$$x \in Z_M \rightarrow (x_1, x_2, \ldots, x_L) = (x \bmod p_1, x \bmod p_2, \ldots, x \bmod p_L).$$

The integers $x \bmod p_i$ are called the *residues of* x *mod* p_i.

To recover to recover an integer x from its set of residues $(x_1, x_2, \ldots, x_L)$, form the quantity $m_i := M/p_i$ for each modulus p_i. Since m_i is relatively prime to the modulus p_i, there exists some $n_i \in Z_{p_i}$ such that $m_i n_i \equiv 1 \bmod p_i$. The inverse mapping is then given by

$$x = \sum_{i=1}^{L} \beta_i x_i = \left(\sum_{i=1}^{L} m_i n_i x_i \right) \bmod M. \tag{2.6}$$

(The mod M operation insures that the result is in Z_M).

2.2.2 The Polynomial CRT

Let F be a field. Recall that by $F[x]$ is meant the ring of polynomials with coefficients in F and that F^N denotes the N-fold cartesian product of F with itself. We are concerned with isomorphisms between the quotient rings $F[x]/\langle x^N - 1 \rangle$ or $F[x]/\langle x^N + 1 \rangle$ and the ring F^N. It is essential that the following condition is met:

Condition 1 The polynomial $x^N \pm 1$ has N distinct linear factors over the base field F. In this case, we have

$$x^N \pm 1 = (x - r_0)(x - r_1) \cdots (x - r_{N-1}).$$

This condition depends on the nature of the field F and the degree N. The Polynomial RNS, for example, is based on the field $F = GF(p)$ for some prime p. Admissible choices for p and N are easily obtained using elementary number theory (see, for example, [31, 60]).

Returning again to the general case of F an arbitrary field, assume that condition 1 is met. Then the principal ideals $\langle x - r_i \rangle$ generated by the distinct linear factors $(x - r_i)$ are pairwise relatively prime. The existence of the isomorphism between $F[x]/\langle x^N \pm 1 \rangle$ and F^N is now guaranteed by the CRT.

To apply the CRT to our problem, let R be $F[x]$. Also, let $I_i = \langle x - r_i \rangle$, $i = 0, 1, \ldots, N-1$ where the r_i are the N distinct roots of the congruence $x^N \pm 1 = 0$. Then $\bigcap_{i=1}^{N} I_i = \langle x^N \pm 1 \rangle$ and hence

$$\frac{F[x]}{\langle x^N \pm 1 \rangle} \cong \frac{F[x]}{\langle x - r_0 \rangle} \oplus \frac{F[x]}{\langle x - r_1 \rangle} \oplus \cdots \oplus \frac{F[x]}{\langle x - r_{N-1} \rangle}.$$

But $F[x]/\langle x - r_i \rangle \cong F$, whence

$$F[x]/\langle x^N \pm 1 \rangle \cong F^N. \tag{2.7}$$

Eq. 2.7 is the fundamental result upon which most fast polynomial multiplication systems are premised. The forward and inverse mappings relating the two isomorphic rings will now be developed. First, let $f(x) = f_0 + f_1 x + \cdots + f_{N-1} x^{N-1}$ be a polynomial in the ring $F[x]/\langle x^N \pm 1 \rangle$. The image $\vec{\phi}$ of $f(x)$ in F^N is given by the canonical homomorphism

$$\vec{\phi} = (f(r_0), f(r_1), \ldots, f(r_{N-1}))^T \in F^N.$$

To recover a polynomial f from its image $\vec{\phi}$, for $k = 0, 1, \ldots, N-1$, define the polynomials $L_k(x)$ by

$$L_{k,0} + L_{k,1} x + \cdots + L_{k,N-1} x^{N-1} = \left(\prod_{j \neq k} (x - r_j) \right) \left(\prod_{j \neq k} (r_k - r_j) \right)^{-1},$$

which are the Lagrange interpolation polynomials over the field F. It immediately follows that $L_k(r_j) = \delta_{kj}$ (where $\delta_{kj} = 0$ if $k \neq j$ and $\delta_{kj} = 1$ if $k = j$), so the inverse of $\vec{\phi}$ is given by

$$\sum_{i=0}^{N-1} f(r_k) L_k(x).$$

The isomorphism allows the product of two polynomials in the ring $F[x]/\langle x^N \pm 1 \rangle$ (a computation which requires $O(N^2)$ F-multiplies) to be computed in the ring F^N (with just $O(N)$ F-multiplies).

2.3 Finite Field Theory

Since the theory of finite fields will be of considerable importance to us, we state some of its fundamental results. Most of the theorems will be without proof, but we provide pseudeo-proofs if necessary for clearer understanding.

A finite field is simply a ring in which all nonzero elements have multiplicative inverses. We already know that the quotient ring Z_p is a finite field if and only if p is prime. A logical question is if there are any other finite fields and if there are, we wish to determine their structure. The following results will show that there are other finite fields and will classify all of them.

Definition 2.8. Let E be a field. Then F is a subfield of E if $F \subset E$ and F itself is a field. Furthermore, E is said to be an extension field of F. This is expressed by $F \leq E$.

Theorem 2.6 (Kronecker's Theorem). Let F be a field and let $p(x)$ be a nonconstant irreducible polynomial in $F[x]$. Then there exists an extension field E of F and an element $\alpha \in E$ such that $p(\alpha) = 0$.

Since $p(x)$ is irreducible, the ideal $(p(x))$ is maximal and it is simple to show that the quotient ring $F[x]/(p(x))$ is an extension field of F. Consider the element $\alpha = x + (p(x))$. Evaluating $p(x)$ at α,

$$p(\alpha) = f_0 + f_1(x + (p(x))) + f_2(x + (p(x)))^2 + \cdots + f_n(x + (p(x)))^n = 0.$$

Definition 2.9. An element α of an extension field E of a field F is algebraic over F if $f(\alpha) = 0$ for some nonzero $f(x) \in F[x]$.

Theorem 2.7. Let E be an extension field of a field F and let $\alpha \in E$ where α is algebraic over F. Then there is an irreducible polynomial $p(x) \in F[x]$ such that $p(\alpha) = 0$. Furthermore, $p(x)$ is uniquely determined up to a constant factor in F and is a polynomial of minimal degree ≥ 1 in $F[x]$ having α as a zero. If $f(\alpha) = 0$ for some $f(x) \in F[x]$, then $p(x)$ divides $f(x)$.

The polynomial $p(x)$ is known as the it irreducible polynomial for α over F. Now, suppose an element α is algebraic over F. Then the above theorem proves that the kernel of the evaluation homomorphism ϕ_α is the principal ideal $(p(x))$. Thus, the quotient ring $F[x]/(p(x))$ is a field and is isomorphic to the image of ϕ_α. In other words,

$$Im(\phi_\alpha) \cong F[x]/(p(x)).$$

The quotient ring is the smallest field containing both F and the element α and will be denoted by $F(\alpha)$.

Definition 2.10. An extension field E of a field F is a simple extension of F if $E = F(\alpha)$ for some $\alpha \in E$.

Theorem 2.8. Let E be a simple extension $F(\alpha)$ of a field F. Let the degree of the irreducible polynomial for α, $p(x)$ be $n \geq 1$. Then E is a vector space over F with basis $\{1, \alpha, \alpha^2, \ldots, a^{n-1}\}$.

What this theorem means is that if E is a simple extension of a field F, where $E = F(\alpha)$, then every element β of E can be written as

$$\beta = c_0 + c_1\alpha + c_2\alpha^2 + \cdots + c_{n-1}\alpha^{n-1}$$

where the c_i are in F and are unique. This is the result we need in order to construct some more finite fields.

Example: The polynomial x^2+1 is irreducible over the reals so that $R[x]/(x^2+1)$ is an extension field of R. Let

$$\alpha = x + (x^2 + 1).$$

Then $R(\alpha)$ is a vector space of dimension two over the reals and consists of all elements of the form $a + b\alpha$ where $a, b \in R$. But $\alpha^2 + 1 = 0$ so that α can be viewed as the imaginary number $j \in C$ and $a + b\alpha$ can be identified with $a + jb \in C$. Thus, $R(\alpha) \cong C$.

Notice that if $F(\alpha)$ is a simple extension of a finite field F and the degree of the irreducible polynomial for α over F is n, then the order of $F(\alpha)$ is p^n where p is the order of F. This is because any element β of $F(\alpha)$ can be written as

$$\beta = c_0 + c_1\alpha + \cdots + c_{n-1}\alpha^{n-1}$$

where $c_i \in F$.

Theorem 2.9. Every finite field contains as a subfield the field Z_p for some prime p.

What this means is that any finite field F can be viewed as an extension of some prime subfield Z_p. This leads immediately to the following.

Corollary 2.1. Let F be a finite field. Then the order of F is p^n for some prime p.

Actually, there is a much stronger result which can be used to classify all finite fields.

Theorem 2.10. Any two fields of order p^n are isomorphic.

This is a very strong result. It says that there is only one finite field of a given order p^n. As a consequence, we will refer to *the* field of order p^n.

The result which allows the design of multiplierless arithmetic units is provided by the following.

Theorem 2.11. If F is a finite field with n elements, the nonzero elements of F form a finite cyclic group of order $n-1$.

What this means is that given any finite field Z_p, there exists an element $\beta \in Z_p$ (called a *primitive element*) whose non-negative powers generate all of the nonzero elements of Z_p.

Example: Let $p = 7$ and $\beta = 3$. Then

$$\begin{aligned} 3^0 &= 1 \bmod 7 \\ 3^1 &= 3 \bmod 7 \\ 3^2 &= 2 \bmod 7 \\ 3^3 &= 6 \bmod 7 \\ 3^4 &= 4 \bmod 7 \\ 3^5 &= 5 \bmod 7. \end{aligned}$$

Notice that if we compute $3^6 \bmod 7$, the result is 1. This suggests the following.

Corollary 2.2. For any prime p and any nonzero integer a,

$$a^{(p-1)} = 1 \bmod p.$$

The previous examples suggests a simple scheme for performing finite field multiplication. Suppose we wish to multiply two nonzero elements x and y of a finite field Z_p. If the nonzero elements are generated by β, there are integers $e_x, e_y \in Z_p$ (called *logarithms*) such that

$$x = \beta^{e_x} \quad \text{and} \quad y = \beta^{e_y}.$$

To compute the product of x and y, simply add their logarithms modulo $p-1$ (because the multiplicative group is of order $p-1$) and look up the result from the list of logarithms. In other words,

$$x \cdot y \bmod p = \beta^{e_x} \cdot \beta^{e_y} = \beta^{(e_x+e_y) \bmod (p-1)} \bmod p. \tag{2.8}$$

Example: Returning again to Z_7, suppose we wish to compute the product of 6 and 4 modulo 7. From the list in the previous example,

$$6 = 3^3 \bmod 7 \quad \text{and} \quad 4 = 3^4 \bmod 7.$$

Then

$$6 \cdot 4 \bmod 7 = 3^3 \cdot 3^4 = 3^{(3+4) \bmod 6} = 3^1,$$

which, from the list, equals 3.

2.4 Associative Algebras

Definition 2.11. An associative algebra over a field F is a pair consisting of a ring $(R, +, \cdot, 0, 1)$ and a vector space R over F such that the underlying set R and the addition and 0 are the same in the ring and vector space, and

$$a(xy) = (ax)y = x(ay)$$

holds for $a \in F$ and $x, y \in R$. If R is finite dimensional over F, then the algebra is said to be finite dimensional.

It is simple to impose the structure of an associative algebra on the rings $F[x]/\langle x^N \pm 1\rangle$ and F^N. Simply maintain the multiplicative and additive structures of both rings and let the field F act on both of the rings with the scalar-vector product defined in

the obvious way. It is trivial to verify that all of the axioms are satisfied and that what results is a pair of associative algebras over F.

Ignoring the multiplicative structure, the vector space (over F) $F[x]/\langle x^N \pm 1\rangle$ is of dimension N with basis

$$\mathcal{B}_1 = \{\vec{v}_0, \vec{v}_1, \ldots, \vec{v}_{N-1}\} := \{1, x, x^2, \ldots, x^{N-1}\}$$

and hence is isomorphic to the F-vector space F^N with canonical basis

$$\mathcal{B}_2 = \{\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_N\}$$

where

$$\vec{e}_i = (0, 0, \ldots, \overset{i}{1}, 0, \ldots, 0).$$

The notion of isomorphic algebras still needs to be clarified.

Definition 2.12. A map of an algebra R into an algebra S over the same field F is an algebra isomorphism if it is both a ring isomorphism and a linear map.

CHAPTER 3
THE INTEGER RNS

3.1 Introduction

The residue number system (RNS) is a system for performing integer arithmetic in many parallel small-wordwidth channels concurrently. It will be seen that the RNS allows for extremely high-speed pipelined arithmetic and is an ideal medium for the VLSI implementation of complex arithmetic operations.

Recall that given a set of pairwise relatively prime integers, we have the following.

Theorem 3.1 (Chinese remainder theorem). Let $\{p_1, p_2, \ldots, p_L\}$ *be a set of pairwise relatively prime integers and define* M *by*

$$M = \prod_{i=1}^{L} p_i.$$

Then

$$Z_M \cong Z_{p_1} \oplus Z_{p_2} \oplus \cdots \oplus Z_{p_L}.$$

Specifically, what the CRT says is that to perform an operation in the larger ring Z_M, the operands can be mapped to the smaller rings Z_{p_i} where the operations are performed concurrently, and the result can be mapped back to Z_M.

The mapping from Z_M to the direct sum ring is simple. Given $X, Y \in Z_M$, map X and Y by

$$X \rightarrow (x_1, x_2, \ldots, x_L)$$

and

$$Y \rightarrow (y_1, y_2, \ldots, y_L)$$

where, for example, $x_i = X \bmod p_i$. The x_i are called the *residues* of X modulo p_i.

The binary composition of X and Y can be performed in the direct sum ring with residue digits given by

$$z_i = x_i * y_i \bmod p_i$$

where "$*$" is either addition, subtraction, or multiplication.

As discussed earlier, the mapping from the direct sum ring to Z_M is as follows. Form the set of integers

$$\{m_1, m_2, \ldots, m_L\}, \quad \text{where} \quad m_i := M/m_i. \tag{3.1}$$

Next, define the integers n_i by

$$m_i n_i = 1 \bmod p_i. \tag{3.2}$$

Then, given an L-tuple of residues $(z_1, z_2, \ldots, z_L)$, its image in Z_M is given by

$$Z = \left(\sum_{i=1}^{L} m_i (n_i z_i \bmod p_i) \right) \bmod M. \tag{3.3}$$

For the CRT to be useful, it is essential that the computation not "overflow the modulus." That is, if we embed integer computations in the ring Z_M, the result should not exceed $M - 1$. If this condition is met, the computation will yield the same result as if it occurred over the integers.

Example: Let the modulus set be given by $3, 5$, and 7. Since the moduli are pairwise relatively prime, and $3 \cdot 5 \cdot 7 = 105$,

$$Z_{105} \cong Z_3 \oplus Z_5 \oplus Z_7.$$

Suppose we wish to use the RNS to compute the product of the integers 21 and 4. Then

$$21 \rightarrow (0,1,0) \quad \text{and} \quad 4 \rightarrow (1,4,4).$$

Now, compute the product pairwise:

$$(0,1,0)(1,4,4) = (0,4,0).$$

We now need the parameters m_i and n_i. From Eq. 3.1,

$$m_1 = 35, \quad m_2 = 21, \quad m_3 = 15.$$

To calculate the n_i, Eq. 3.2 says we need $m_i n_i \equiv 1 \bmod p_i$ which gives

$$n_1 = 2, \quad n_2 = 1, \quad n_3 = 1.$$

Finally, use Eq. 3.3 to obtain

$$4 \cdot 21 = (35 \cdot (2 \cdot 0 \bmod 3) + 21 \cdot (1 \cdot 4 \bmod 5) + 15 \cdot (1 \cdot 0 \bmod 7)) \bmod 105 = 84.$$

Another method that can be used to convert from residue to integer is the *mixed radix conversion* (MRC) algorithm [67]. Associated with an integer X and a set of moduli $\{p_1, p_2, \ldots, p_L\}$ is a set of mixed radix digits $\{a_1, a_2, \ldots, a_L\}$ such that

$$X = \sum_{k=2}^{L} a_k \left(\prod_{j=1}^{k-1} p_j \right) + a_1.$$

(Note the similarity to a conventional fixed radix representation.) The mixed radix digits can be computed from the residues according to the recursion

$$\begin{array}{lcl} X^{(0)} & = & X \\ a_1 & = & x_1 \\ X^{(i)} & = & p_{i-1}^{-1}(X^{(i-1)} - a_{i-1}) \bmod p_i \\ a_i & = & X^{(i)} \bmod p_i \end{array}$$

The mixed radix conversion algorithm is most useful in applications such as base extension or sign detection. The MRC, however, is more hardware-intensive and slower than some of the elegant CRT approximation techniques we will describe in the coming sections, and hence will not be discussed further.

3.2 Signed RNS

The development thus far has not considered how to deal with signed arithmetic. Recall that the RNS is based on some ring Z_M (assume M is odd) in which the additive inverse of a number $0 \leq X \leq (M/2 - 1)$ is given by

$$-X = M - X.$$

In this case, map an integer $-X$ to its set of residues

$$-X \rightarrow (p_1 - (X \bmod p_1), p_2 - (X \bmod p_2), \ldots, p_L - (X \bmod p_L)).$$

This provision is easily built into the tables which comprise the input conversion units.

3.3 An RNS System

A RNS system can be broken into three basic subsystems:

1. Input conversion subsystem
2. QRNS Computational Units
3. CRT (output conversion) subsystem

Briefly, the role of the input conversion subsystem is to deliver data in RNS format to the parallel RNS computational units. The data is processed in the RNS by the

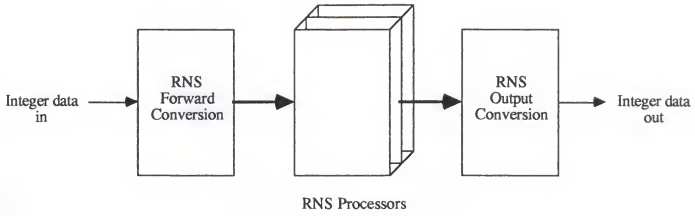


Figure 3.1. The basic components of an RNS system

computational units and then converted from RNS format back to complex integer by the output conversion subsystem. The situation is as depicted in Fig. 3.1

Each of these subsystems will now be examined independently.

3.3.1 RNS Input Conversion

Assume that our RNS is based on a set of moduli

$$P = \{p_1, p_2, \ldots, p_L\}.$$

The purpose of input conversion is to convert an integer X to its set of residues corresponding to each modulus. Symbolically, this is represented by

$$X \rightarrow ((x_1, x_2, \ldots, x_L))$$

where

$$x_i := X \bmod p_i.$$

The computation of x_i is fairly straightforward. Simply present X a table and the table returns $X \bmod p_i$. If X is of appreciable magnitude, the address space of the table can become prohibitively large. In such a case, the modular reduction can be accomplished by spreading the modular reduction across two tables. For example, suppose X is 16 bits in size. Decompose X into two 8-bit words, X_{HI} and X_{LO}, *i.e.*,

$$X = 2^8 X_{HI} + X_{LO}.$$

Then

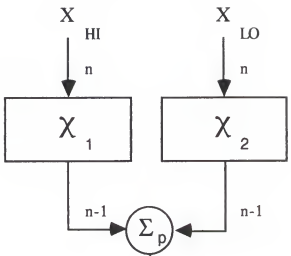
$$X \bmod p_i = \left((2^8 X_{HI}) \bmod p_i + X_{LO} \bmod p_i \right) \bmod p_i.$$

Now, two smaller tables are used for the modular reduction of $2^8 X_{HI}$ and X_{LO}. Reducing the 16-bit integer X modulo p_i is performed with two tables and a modulo-p_i adder. The saving in table space is enormous; two tables with 2^8 addresses are much smaller than one table with 2^{16} addresses. The only additional expense is the inclusion of one extra level of modulo-p_i addition, which is easily pipelined to maintain the throughput of the input converter. Thus, the realization of ϕ_{p_i} where p_i is limited to eight bits and the input data is 16-bits consists of two 256×8 tables and one modulo-p_i adders. Schematically, this is shown in Fig. 3.2

3.3.2 The RNS Computational Unit

For the present discussion, the only functions that need to be performed by an RNS ALU are addition and multiplication. In what follows, it is assumed that the integer X has already been mapped by ϕ_{p_i} to its residue x_i.

Addition in the RNS is performed is performed modulo-p_i which can be done in one of two ways. The first is to use a binary adder followed by a table for modular reduction. The second, and preferred, is to use a modulo-p_i adder which consists



$$X_1 = 2^8 X_{HI} \bmod p$$
$$X_2 = X_{LO} \bmod p$$

$$Z = (2^8 X_{HI} \bmod p + X_{LO} \bmod p) \bmod p$$

Figure 3.2. RNS forward conversion element.

of a pair of binary adders and some glue logic. We will discuss multiplication in a following chapter.

Mod-p Adder

This section shows how to build a mod-p adder from standard binary adders. Let p be an integer such that

$$0 < p \leq 2^n - 1$$

and suppose we use n-bit binary adders configured as shown in Fig. 3.3. Denote by Σ_A and O_A the output and the overflow of the adder A, respectively, and likewise for adder B. It will be assumed that the summands, x and y are both less than p so that

$$x + y \leq 2(p - 1).$$

There are two cases to consider:

Case 1 $0 < p \leq 2^{n-1} - 1$

Case 2 $2^{n-1} \leq p \leq 2^n - 1$

Case 1

There are two possibilities for the uncorrected sum of x and y:

a) $0 \leq x + y \leq p - 1$

b) $p \leq x + y \leq 2(p - 1)$

a) Adder A does not overflow, so $O_A = 0$ and $\Sigma_A = x + y$. Then

$$\Sigma_B = (\Sigma_A + 2^n - p) \bmod 2^n = (x + y + 2^n - p) \bmod 2^n.$$

But

$$x + y + 2^n - p \leq p - 1 + 2^n - p = 2^n - 1$$

so that O_B=0 and Σ_B is incorrect. The correct result is given by Σ_A.

b) We have $2(p-1) \leq 2^n - 1$ so that adder A still does not overflow, and $O_A = 0$ and $\Sigma_A = x + y$. Then

$$\Sigma_B = (\Sigma_A + 2^n - p) \bmod 2^n = (x + y + 2^n - p) \bmod 2^n.$$

But it is assumed that $p \leq x + y$ so that

$$2^n \leq x + y + 2^n - p.$$

Thus, adder B overflows, $O_B = 1$, and

$$\Sigma_B = (x + y + 2^n - p) \bmod 2^n = x + y - p,$$

which is the correct result. Hence, for Case 1, if neither adder overflows, the correct result is given by adder A and if either adder overflows, the correct result is given by adder B.

Case 2

There are three possibilities for the uncorrected sum, $x + y$:

a) $0 \leq x + y \leq p - 1$

b) $p \leq x + y \leq 2^n - 1$

c) $2^n \leq x + y$

a) Adder A does not overflow, so $O_A = 0$ and $\Sigma_A = x + y$. Then

$$\Sigma_B = (\Sigma_A + 2^n - p) \bmod 2^n = (x + y + 2^n - p) \bmod 2^n.$$

But

$$x + y + 2^n - p \leq p - 1 + 2^n - p = 2^n - 1$$

so that O_B=0 and Σ_B is incorrect. The correct result is given by Σ_A.

b) $2(p - 1) \leq 2^n - 1$ so that adder A still does not overflow, and $O_A = 0$ and $\Sigma_A = x + y$. Then

$$\Sigma_B = (\Sigma_A + 2^n - p) \bmod 2^n = (x + y + 2^n - p) \bmod 2^n.$$

But it is assumed that $p \leq x + y$ so that

$$2^n \leq x + y + 2^n - p.$$

Thus, adder B overflows, $O_B = 1$, and

$$\Sigma_B = (x + y + 2^n - p) \bmod 2^n = x + y - p,$$

so the correct result is given by Σ_B.

c) In this case, adder A overflows so that $O_A = 1$ and

$$\Sigma_A = x + y - 2^n.$$

Then

$$\Sigma_B = (\Sigma_A + 2^n - p) \bmod 2^n.$$

But

$$\Sigma_A + 2^n - p = x + y - 2^n + 2^n - p = x + y - p.$$

Also,

$$x + y - p \leq 2(p-1) - p = p - 2 \leq 2^n$$

so that adder B does not overflow, *i.e.*, $O_B = 0$ and

$$\Sigma_B = x + y - p,$$

which is the correct result. As for case 1, if neither adder overflows, the correct result is given by Σ_A and if either adder overflows, the correct result is given by Σ_B. The selection of the adder outputs is simply the logical *or* of the overflow bits.

3.3.3 Output Conversion

The purpose of output conversion is to recover an integer from its set of residues. As we mentioned, there are several techniques to recover an integer from its set of residues, including the Chinese remainder theorem, mixed-radix conversion, and the core function method. Each of these techniques carries its own set of advantages and disadvantages, but a complete discussion would be too lengthy. We prefer to use the CRT method because it leads to some approximate conversion methods which are extremely fast, simple, and small when implemented in hardware.

3.3.4 Efficient CRT implementation

There have been many papers on residue-to-decimal conversion and scaling (see [78, 79, 36, 37, 38, 65, 29], for example). Most of these papers restrict the class of permissible moduli for the RNS. While this results in an efficient design, it does not allow conversion for the quadratic RNS (QRNS) or for other systems not based on the admissible moduli.

Other algorithms have been developed that allow a broader class of modulus sets but require large modulo M adders and a great number of ROMs [36, 37]. As such,

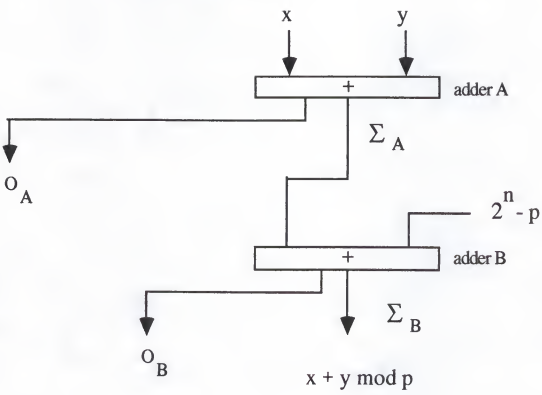


Figure 3.3. Mod-p adder architecture

these methods result in an expensive and complex design. A more desirable approach is to search for a method that uses standard *binary* hardware and does not require any expensive custom parts.

From Eq. 3.3, recall that

$$X = \left(\sum_{i=1}^{L} m_i (n_i x_i \bmod p_i) \right) \bmod M. \tag{3.4}$$

The primary limitation in hardware realizations of this equation is the need for the large modulo M addition at the end. For an RNS design to have any utility, there must be enough dynamic range to accomodate the desired computations. In the FFTAP, for example, the dynamic range, M, needs to be roughly 40 bits. A modulo M adder where M is about 40 bits in size would be extremely complex, costly, and slow. The problem is compounded by the fact that we need to perform eight simultaneous *complex* CRT conversions, requiring a total of sixteen such CRT engines.

Instead, an approximate CRT engine can be built if one realizes that one of the primary roles of the CRT engine is to accommodate scaling. That is, an intermediate result is converted to integer from its RNS representation, scaled, and returned to the RNS for further processing. As we will see, scaling is crucial in the FFTAP design: at each stage of the radix-8 FFT, an additional 24 bits of dynamic range are accumulated on top of the 16-bit input precision. To avoid loss of significance in the FFTAP, the 40-bit accumulated intermediate results are scaled back to 16-bit integers between FFT stages.

To develop the scaling CRT algorithm, define a_i by

$$a_i := m_i (n_i x_i \bmod p_i).$$

Then

$$X = \sum_{i=1}^{L} a_i \bmod M.$$

Given a scale factor d, where $1 \leq d \leq M$, approximate the a_i/d by the real numbers α_i and M/d by the real number μ such that

$$|a_i/d - \alpha_i| \leq \epsilon < \frac{M}{dL}$$

and

$$|M/d - \mu| \leq \delta < \frac{M}{dL} - \epsilon.$$

We call μ the *reduced modulus.*

If y is defined by

$$y = \left(\sum_{i=1}^{L} \alpha_i\right) \bmod \mu$$

then the error between y and X/d can be shown [29] to lie either in the range

$$|x/d - y| < L(\epsilon + \delta)$$

or in the range

$$\min(M/d, \mu) - L(\epsilon + \delta) < |x/d - y| < \max(M/d, \mu).$$

The elements of Z_M whose approximation errors fall into the latter error range are said to lie in the *catastrophic error band.* We will now show how to avoid catastrophic errors.

It can be shown [29] that the elements of the catastrophic error band satisfy either

$$0 \leq X \leq dL(\epsilon + \delta)$$

or

$$M - dL(\epsilon + \delta) < X < M.$$

In a signed RNS, the error band is centered around $M/2$ so that elements of the catastrophic error band satisfy

$$M/2 - dL(\epsilon + \delta) < X < M/2 + dL(\epsilon + \delta).$$

If the range of computations does not overlap the catastrophic error band, the corresponding errors will be less than $L(\epsilon + \delta)$ in magnitude. Thus, if the dynamic range of our computations is given by r, we must ensure that

$$r \leq M/2 - dL(\epsilon + \delta).$$

Recall that $\epsilon < M/(dL)$ and $\delta < M/(dL) - \epsilon$ so that the reduced modulus may be any number μ in the range

$$\frac{M}{d}\left(1 - \frac{1}{2L}\right) + \frac{r}{dL} + \epsilon \leq \mu \leq \frac{M}{d}\left(1 + \frac{1}{2L}\right) - \frac{r}{dL} - \epsilon.$$

For practical CRT implementation, choose a scaling factor d so that $M = d2^k$. In this case, 2^k is the reduced modulus. In other words, we wish to scale intermediate results so they are bounded by 2^k. Next, define the numbers α_i by

$$\alpha_i := [[m_i(n_i x_i \bmod p_i)/d]]$$

where $[[\cdot]]$ indicates truncation. If the scaled result is generated by

$$X_s = \sum_{i=1}^{L} \alpha_i \bmod 2^k, \tag{3.5}$$

then $\epsilon = 1/2$, $\delta = 0$, and catastrophic errors can be avoided if k is chosen such that

$$2^k\left(1 - \frac{2}{L}\right) + \frac{r}{dL} + \frac{1}{2} \leq 2^k \leq 2^k\left(1 + \frac{2}{L}\right) - \frac{r}{dL} - \frac{1}{2}. \tag{3.6}$$

In this case,

$$|X/d - X_s| < L/2.$$

Typical values for L are less than eight, so that the error incurred in this approximation is insignificant relative to the dynamic range. Equation 3.5 can be realized with nothing more than binary adders and small tables, in contrast with the larger tables and large custom modular adders required by Eq. 3.4. A scaling CRT engine is shown in Fig. 3.4

Recall that the signed RNS maps positive to the lower half of the dynamic range and negative numbers to the upper half. Typically, this represents a difficulty in interfacing with conventional systems since the magnitude of a negative number is given by $M - |x|$. The scaled CRT, however, uses a reduced dynamic range which is a power of 2. Thus, the negative numbers are given by $2^n - |x|$, which is the standard two's-complement form for a negative number x. An important side-effect of the scaling CRT, then, is that the output data has been "magically" transformed to two's complement form.

Another approach to CRT implementation uses distributed arithmetic. We introduce here an algorithm for residue-to-decimal conversion based on distributed arithmetic which allows a very simple and efficient hardware realization. The approach we will take is based on the idea of accomplishing modular reduction by ignoring integer overflow in binary adders. This has been done previously by Soderstrand *et al.* [68], but our method is more accurate and leads to a simple scheme for scaling.

Distributed arithmetic has been recognized as a useful technology for digital filtering [54]. Jenkins and Leon [38] considered the use of distributed arithmetic in residue-to-decimal conversion, but their design depended on the use of modulo M adders. The method is similar in spirit to [38, 65], but results in a reduction in hardware, cost, and complexity over earlier designs.

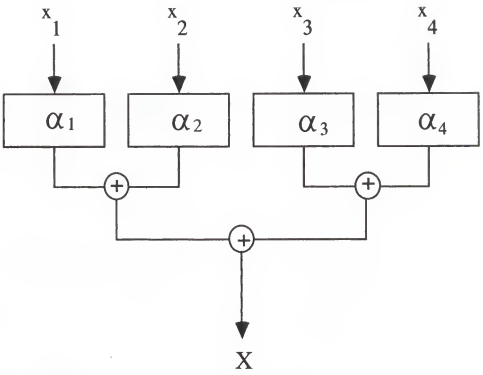


Figure 3.4. Simplified scaling CRT engine ($\mu = 2^{16}$)

Consider forming the *scaled* result X_s defined by

$$X_s = \frac{1}{M} X.$$

Then X_s is expressed by

$$X_s = \sum_{i=1}^{L} \frac{1}{M} m_i n_i x_i - q = \sum_{i=1}^{L} \frac{1}{p_i} n_i x_i - q.$$

It is clear that X_s lies in the range $[0, 1)$. Also, denote by β_i the quantity $(1/p_i)n_i$. It is also obvious that β_i lies in the range $[0, 1)$ (this will be useful later). The final form of the equation for X_s is

$$X_s = \sum_{i=1}^{L} \beta_i x_i - q \tag{3.7}$$

and since $X_s \in [0, 1)$, the subtraction of q can be implemented by discarding any integer bits.

As it stands, however, there is no simple realization of Eq. (3.7) since the β_i may not have any finite-wordlength representation. We would like to be able to calculate X by forming some quantity $\hat{X}_s$, scaling by M, and ignoring any fractional bits. In other words, we want $\hat{X}_s$ such that $\lfloor M\hat{X}_s \rfloor = X$. Thus, it would be desirable if the β_i could be approximated by some finite wordlength quantity $\hat{\beta}_i$ such that the cumulative approximation error in the equation

$$\hat{X}_s = \sum_{i=1}^{L} \hat{\beta}_i x_i - q \tag{3.8}$$

is removed by truncation when $\hat{X}_s$ is scaled by M. Therefore, we require that $\hat{\beta}_i = \beta_i + e_i$, where $e_i \geq 0$.

The only problem remaining is how to accurately represent the β_i in fixed-wordlength. What we can do is truncate the fractional representation of β_i to r fractional bits

and add 2^{-r} so that $\hat{\beta}_i \geq \beta_i$. Consider the r-bit representation

$$\hat{\beta}_i = \beta_i - 2^{-r}\text{frac}(2^r\beta_i) + 2^{-r} = \lceil 2^r\beta_i \rceil 2^{-r}. \tag{3.9}$$

Now, calculate X_s by

$$\hat{X}_s = \sum_{i=1}^{L} \hat{\beta}_i x_i - q.$$

It can be seen that the error $e_i = \hat{\beta}_i - \beta_i$ in approximating β_i by $\hat{\beta}_i$ lies in the range $[0, 2^{-r})$ so that the total error e can be calculated in the following manner.

First,

$$\hat{X}_s = \sum_{i=1}^{L} \hat{\beta}_i x_i - q = \sum_{i=1}^{L} (\beta_i + e_i)x_i - q.$$

Now, with $\hat{X}_s$ given as above, the error in approximating X_s by $\hat{X}_s$ is given by

$$e = \hat{X}_s - X_s = \sum_{i=1}^{L} (\beta_i + e_i)x_i - q - \sum_{i=1}^{L} \beta_i x_i + q = \sum_{i=1}^{L} e_i x_i, \tag{3.10}$$

and since $e_i \in [0, 2^{-r})$ and $x_i \in [0, p_i)$,

$$0 \leq e < \sum_{i=1}^{L} 2^{-r} x_i < 2^{-r} \cdot \sum_{i=1}^{L} (p_i - 1).$$

The ultimate objective is to pass $\hat{X}_s$ back to the main processor and scale by M and truncate to integer. In this case,

$$\hat{X} = \lfloor M\hat{X}_s \rfloor = \lfloor MX_s + Me \rfloor.$$

If we force $e < 1/M$, then $Me < 1$, and

$$\hat{X} = \lfloor MX_s + Me \rfloor = \lfloor X + Me \rfloor = X.$$

Thus, if we require the worst case,

$$2^{-r} \sum_{i=1}^{L} (p_i - 1) < 1/M,$$

our scaled result will be correct. In other words, the number of bits needed to approximate β_i unambiguously is given by

$$\log_2 \left(M \cdot \sum_{i=1}^{L} (p_i - 1) \right) < r. \tag{3.11}$$

Distributed Arithmetic

Recall that the calculate the scaled result $\hat{X}_s$ is given by

$$\hat{X}_s = \sum_{i=1}^{L} \hat{\beta}_i x_i - q$$

and in practice, that we can compute the quantity

$$\Omega := \sum_{i=1}^{L} \hat{\beta}_i x_i \tag{3.12}$$

and discard any integer bits.

Assume that we have calculated r, the number of bits needed to represent $\hat{\beta}_i$ by Eq. (3.11). Since $\hat{\beta}_i < 1$, there are r fractional bits and no integer bits. Furthermore, assume that all of the moduli $p_1, p_2, \ldots, p_L$ are b bits in length or less.

Let the binary representation of x_i be given by

$$x_i \sim x_i^{b-1} x_i^{b-2} \cdots x_i^0$$

where $x_i^j = 0$ *or* 1, $j = 0, 1, \ldots, b-1$. Then

$$x_i = \sum_{j=0}^{b-1} x_i^j 2^j. \tag{3.13}$$

With Ω given as above, notice that the $\hat{\beta}_i$ are precomputed and the x_i can be regarded as "incoming data." With this in mind, substitute Eq. (3.13) into Eq. (3.12) to obtain

$$\Omega = \sum_{i=1}^{L} \hat{\beta}_i \sum_{j=0}^{b-1} x_i^j 2^j. \tag{3.14}$$

Now, interchange the order of summation so that

$$\Omega = \sum_{j=0}^{b-1} 2^j \sum_{i=1}^{L} \hat{\beta}_i x_i^j. \tag{3.15}$$

The second summation in the above equation can be interpreted as a weighted sum of the $\hat{\beta}_i$ where the "weights" are the j-th bits of words $x_1, x_2, \ldots, x_L$. If we let

$$\phi(b_1, b_2, \ldots, b_L) := \sum_{i=1}^{L} b_i \hat{\beta}_i, \tag{3.16}$$

then ϕ can be computed by table look-up. The word $b_1 b_2 \cdots b_L$ is used as an address to a ROM of height 2^L and the precomputed sum in Eq. (3.16) is stored at that address. Thus, substituting Eq. (3.16) into Eq. (3.15), we obtain

$$\Omega = \sum_{j=0}^{b-1} 2^j \phi(x_1^j, x_2^j, \ldots, x_L^j). \tag{3.17}$$

Equation (3.17) can be interpreted as a series of table look-ups followed by shifted additions. The table space is relatively small: for the case of three 7-bit moduli, three address lines are needed for the eight possible addresses, and the width of the ROM is roughly 28 bits. It is possible to break the width into several "narrower" tables.

Practical Aspects

Equation (3.17) tells us that Ω is calculated by addressing a ROM with the parallel combination of the $j-$th bits of $x_1, x_2, \ldots, x_L$, shifting, accumulating, then repeating the process b times where b is the wordlength of the largest modulus.

Equation (3.12) says that integer bits are not significant in the calculation of Ω. This suggests the following scheme:

- Determine r, the number of bits needed to represent the $\hat{\beta}_i$ unsing Eq. (3.11). Thus, there are r fractional bits and n integer bits.

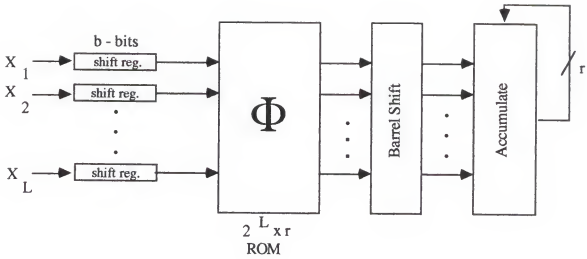


Figure 3.5. Block diagram of DA-CRT

- With ϕ given by Eq. (3.16), it is clear that although the true value of ϕ may have an integer part, these higher bits will be discarded in the calculation of Ω. Thus, the stored values of ϕ need only have r bits total wordwidth.

- The multiplication by 2^j in Eq. (3.17) will shift some bits of ϕ into integer positions. These can be discarded. Thus, the shift register only needs to be of length r with bits shifted out of the register to the left ignored.

- Addition of the shifted values of ϕ is simple. The position of the binary point never changes and there is no need for carrying into integer positions. Thus, a r-bit binary adder will suffice.

- The latency of the converter will be minimized if many small moduli are used rather than two or three large moduli. Additionally, the width of the table,

given by Eq.(3.11) will be smaller since the component introduced by the term $\sum(p_i - 1)$ will be smaller.

CHAPTER 4
THE QUADRATIC RNS

We need to emulate the complex numbers using modular arithmetic. Early attempts at this simply considered performing complex operations over the RNS. Specifically, if $A + jB$ and $C + jD$ were complex integers, they were mapped to their sets of residues as

$$A + jB \rightarrow (a_1 + jb_1, \ldots, a_L + jb_L)$$

and

$$C + jD \rightarrow (c_1 + jd_1, \ldots, c_L + jd_L),$$

the componentwise product was given by

$$(a_i + jb_i)(c_i + jd_i) = (a_i d_i - b_i d_i) \bmod p_i + j(b_i c_i + a_i d_i) \bmod p_i,$$

and the real and imaginary parts recovered separately by the CRT. This scheme is known as the complex RNS (CRNS) and suffers from the same problem as conventional complex arithmetic: a complex multiply operation requires four real products and two real additions. The *quadratic residue number system* (QRNS) mitigates this condition and will now be explained.

To begin, we need the concept of *polynomial quotient rings*. Let $F[x]$ be the ring of polynomials in an indeterminate x over the field F and $p(x)$ a polynomial in $F[x]$ of degree n. By the symbol

$$F[x]/(p(x))$$

is meant the set of polynomials of degree at most $n-1$ with coefficients in F. Addition of these polynomials is performed as usual, but multiplication is performed in a different way. Specifically, two polynomials $f(x)$ and $g(x)$ are multiplied and the degree of the product is reduced to $n-1$ or less by making the substitution $p(x) = 0$.

For example, let $F = R$, the real numbers, and let $p(x) = x^2 + 1$. Suppose we wish to multiply $A + Bx$ and $C + Dx$. We first obtain

$$(A + Bx)(C + Dx) = BDx^2 + (AD + BC)x + AC$$

and substitute $x^2 + 1 = 0$, or $x^2 = -1$ to get

$$(A + Bx)(C + Dx) = (AD + BC)x + (AC - BD).$$

This should look suspiciously similar to the familiar multiplication of the complex numbers.

We can now state a simple version of the Chinese remainder theorem applied to polynomial rings.

Theorem 4.1 (Polynomial CRT). Let F be a field and a and b distinct elements of F. Then

$$\frac{F[x]}{(x-a)(x-b)} \cong F \oplus F.$$

Moving on, let p be a prime of the form $p = 4k + 1$ for some integer k. We make use of a classic result from number theory to derive the QRNS.

Theorem 4.2. Let p be a prime of the form $p = 4k + 1$. Then the equation

$$x^2 \equiv -1 \bmod p$$

has two solutions over the ring Z_p. Furthermore, these two solutions are additive and multiplicative inverses of one another.

If $p = 4k + 1$ (henceforth called a *QRNS prime*), the polynomial $x^2 + 1$ is reducible over Z_p and can be factored as

$$x^2 + 1 = (x - \hat{\jmath})(x + \hat{\jmath}).$$

Then the CRT applied to polynomial rings says that

$$Z_p[x]/(x^2 + 1) \cong Z_p \oplus Z_p := (Z_p)^2.$$

The forward mapping ϕ_p from $Z_p[x]/(x^2 + 1)$ to $Z_p \oplus Z_p$ is given by

$$\phi_p(A + Bx) = ((A + \hat{\jmath}B \bmod p), (A - \hat{\jmath}B \bmod p)) \tag{4.1}$$

and the inverse mapping is given by

$$\phi_p^{-1}((z, z^*)) = (2^{-1}(z + z^*) \bmod p) + j((2\hat{\jmath})^{-1}(z - z^*) \bmod p) \tag{4.2}$$

where $(\cdot)^{-1}$ indicates the multiplicative inverse modulo p.

Here is how to use the QRNS to multiply complex numbers. Let $A + jB$ and $C + jD$ be complex. Apply ϕ_p to both complex numbers to map them to the QRNS:

$$\phi_p(A + jB) = (w, w^*) \quad \text{and} \quad \phi_p(C + jD) = (x, x^*)$$

and compute the QRNS product pairwise, *i.e.*,

$$(z, z^*) = (w, w^*)(x, x^*) = (wx \bmod p, w^*x^* \bmod p).$$

The complex product $E + jF$ is recovered by applying inverse mapping ϕ_p^{-1} to the QRNS pair (z, z^*):

$$E + jF = \phi_p^{-1}((z, z^*)).$$

Example: Let $p = 13$ (a valid QRNS prime). Then $j = 5$ since $5^2 = 25 \equiv -1 \bmod 13$. Also, $2^{-1} = 7$ and $(2j)^{-1} = 4$. Suppose we wish to multiply $2 + j1$ and $3 + j2$. First,

$$\phi_p(2 + j1) = (2 + 5 \cdot 1, 2 + 8 \cdot 1) = (7, 10)$$

and

$$\phi_p(3 + j2) = (3 + 5 \cdot 2, 3 + 8 \cdot 2) = (0, 6).$$

Next, compute the product pairwise:

$$(7, 10)(0, 6) = (0, 8)$$

and apply ϕ_p^{-1} to $(0, 8)$ to obtain

$$\phi_p^{-1}((0, 8)) = (7 \cdot (0 + 8) \bmod 13) + j(4 \cdot (0 - 8) \bmod 13) = 4 + j7,$$

which is the correct result.

4.1 Multiple Modulus QRNS

What we have just examined is a QRNS based on one modulus, p. Such a system is of limited use since the dynamic range is limited by p. A more useful scheme would be to base a QRNS on a modulus set of multiple QRNS primes. This is simply a logical extension of the RNS to the single modulus QRNS. To this end, let $\{p_1, p_2, \ldots, p_L\}$ be a set of QRNS primes. If $M = \prod_{i=1}^{L} p_i$, we then have the isomorphism

$$\frac{Z_M[x]}{(x^2 + 1)} \cong (Z_{p_1})^2 \oplus (Z_{p_2})^2 \oplus \cdots \oplus (Z_{p_L})^2.$$

In this case, the mapping ϕ_M represents the composite mapping

$$\phi_M := (\phi_{p_1}, \phi_{p_2}, \ldots, \phi_{p_L})$$

and similarly for the inverse mapping ϕ_M^{-1}.

Example: Let $p_1 = 5, p_2 = 13$, and $p_3 = 17$. Then $M = 5 \cdot 13 \cdot 17 = 1105$. To multiply $3 + j4$ and $10 + j5$, apply ϕ_M:

$$\phi_M(3 + j4) = ((1,0),(10,9),(4,2))$$

and

$$\phi_M(10 + j5) = ((0,0),(9,11),(7,13)) .$$

Compute the products pairwise modulo each p_i:

$$((1,0),(10,9),(4,2)) \cdot ((0,0),(9,11),(7,13)) = ((0,0),(12,8),(11,9)) .$$

Apply ϕ_M^{-1} to $((0,0),(12,8),(11,9))$:

$$\phi_M^{-1}((\cdot)) = \left(\phi_5^{-1}(0,0), \phi_1^{-1}3(12,8), \phi_{17}^{-1}(11,9)\right) = (0 + j0, 10 + j3, 10 + j4).$$

Finally, apply the integer CRT to the real and imaginary parts:

$$(0,10,10) \rightarrow 10 \quad \text{and} \quad (0,3,4) \rightarrow 55$$

which give the correct result, $10 + j55$.

4.2 A QRNS System

As with the RNS, a QRNS system can be broken into three basic subsystems:

1. Input conversion subsystem
2. QRNS Computational Units
3. CRT (output conversion) subsystem

The role of the input conversion subsystem is to deliver data in QRNS format to the parallel QRNS computational units. The data is processed in the QRNS by the computational units and then converted from QRNS format back to complex integer by the output conversion subsystem. Each of these subsystems will now be examined independently. We consider the case where the input data is sixteen bits wide and we wish to use eight-bit hardware.

4.2.1 QRNS Input Conversion

Assume that our QRNS is based on a set of QRNS moduli

$$P = \{p_1, p_2, \ldots, p_L\}.$$

The purpose of input conversion is to convert a complex integer $A + jB$ to its set of QRNS pairs corresponding to each modulus. Symbolically, this is represented by

$$\phi_M(A + jB) = ((z_1, z_1^*), (z_2, z_2^*), \ldots, (z_L, z_L^*))$$

where ϕ_M represents the composite mapping

$$\phi_M := (\phi_{p_1}, \phi_{p_2}, \ldots, \phi_{p_L}).$$

Recall from Eq. 4.1 that ϕ_{p_i} is the mapping from $Z_M[x]/(x^2 + 1)$ to $(Z_{p_i})^2$ given by

$$\phi_{p_i}(A + jB) = (z_i, z_i^*) := ((A + \hat{j}B \bmod p_i), (A - \hat{j}B \bmod p_i)).$$

The computation of z_i is fairly straightforward. Simply present A to one table and B to another table. The first table returns A mod p_i and the second table returns $\hat{j}B$ mod p_i. The two results are then added using a modulo p_i adder (which is simple to design since p_i is typically eight bits or less in magnitude). This is possible because of the following property of modular reduction:

$$(x + y) \bmod p = (x \bmod p + y \bmod p) \bmod p.$$

The procedure for computing z_i^* is structurally identical, only the modulo-p_i addition is replaced with modulo-p_i subtraction.

If A and B are of appreciable magnitude, the address space of the table can become prohibitively large. In such a case, the modular reduction can be accomplished by spreading the modular reduction across two tables. For example, suppose A is 16 bits in size. Decompose A into two 8-bit words, A_{HI} and A_{LO}, *i.e.*,

$$A = 2^8 A_{HI} + A_{LO}.$$

Then

$$A \bmod p_i = \left((2^8 A_{HI}) \bmod p_i + A_{LO} \bmod p_i\right) \bmod p_i.$$

Now, two smaller tables are used for the modular reduction of $2^8 A_{HI}$ and A_{LO}. Reducing the 16-bit integer A modulo p_i is performed with two tables and a modulo-p_i adder. The same holds for B. Now that A and B have been reduced modulo p_i, the rest of the mapping ϕ_{p_i} proceeds as before. The saving in table space is enormous; two tables with 2^8 addresses are much smaller than one table with 2^{16} addresses. The only additional expense is the inclusion of one extra level of modulo-p_i addition, which is easily pipelined to maintain the throughput of the input converter. Thus, the realization of ϕ_{p_i} where p_i is limited to eight bits and the input data is 16-bits consists of four 256×8 tables and four modulo-p_i adders. Schematically, this is shown in Fig. 4.1

It is important to realize that the QRNS owes its compactness and speed to both the Chinese remainder theorem and the cyclic nature of finite fields. The CRT allows large-wordwidth computations to be decomposed into concurrent small-wordwidth operations. The CRT applied to polynomial rings also decouples the real and imaginary channels to greatly simplify complex arithmetic. The fact that finite fields have

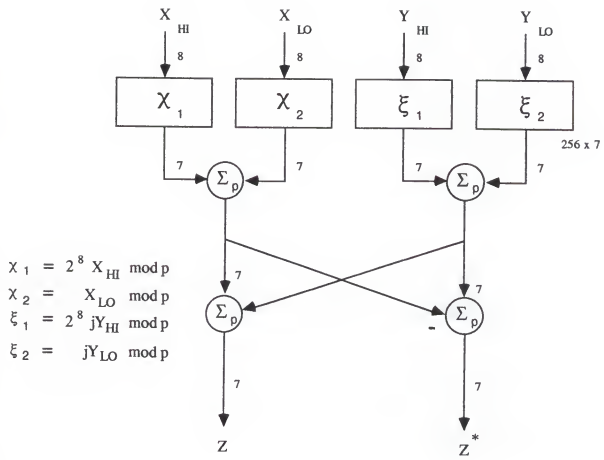


Figure 4.1. QRNS forward conversion element.

a cyclic multiplicative group facilitates the design of multiplier-less arithmetic units. Thus, the QRNS lends itself well to VLSI signal processing hardware design.

4.2.2 The QRNS Computational Unit

For the present discussion, the only functions that need to be performed by a QRNS ALU are addition and multiplication. Subsequent discussions will find the QRNS computational units performing more elaborate computations such as eight point DFTs. These larger computational units, however, are based on the primitive operations of addition and multiplication. In what follows, it is assumed that the complex data has already been mapped by ϕ_{p_i} to its QRNS pair $(z_i, z_i^*) \in (Z_{p_i})^2$.

Addition in the QRNS is performed as a pair of additions; one addition is performed in each of the two subchannels. The addition is performed modulo-p_i which can be performed in one of two ways. The first is to use a binary adder followed by a table for modular reduction. The second, and preferred, is to use a modulo-p_i adder which consists of a pair of binary adders and some glue logic.

As discussed earlier, multiplication can be realized by index addition in the QRNS. Suppose we wish to compute the product of (w_i, w_i^*) and (x_i, x_i^*). Then there exists a primitive element β_i and exponents $e_{w_i}, e_{w_i^*}, e_{x_i}$, and $e_{x_i^*}$ such that

$$(w_i, w_i^*) = (\beta^{e_{w_i}}, \beta^{e_{w_i^*}})$$

and

$$(x_i, x_i^*) = (\beta^{e_{x_i}}, \beta^{e_{x_i^*}}).$$

Then, according to Eq. 2.8 the exponents are added modulo-$(p_i - 1)$ in each subchannel and tables are used to look up the QRNS pair corresponding to the new pair of exponents. Typically, we will not be performing general-purpose multiplications

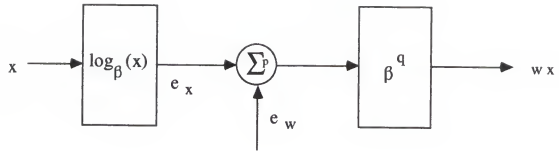


Figure 4.2. QRNS multiplier unit using index addition

in the QRNS since most DSP algorithms involve "scaling" in which one of the multiplicands is known *a priori*. The QRNS exponent pair corresponding to this data is typically stored in memory, hence eliminating one look-up. Thus, a QRNS multiplication unit consists of two tables and one modulo- $(p_i - 1)$ adder per subchannel for a total of four tables and two modulo-$(p_i - 1)$ adders per modulus. Such a unit is depicted in Fig. 4.2. The sequence of table look-ups and modular additions is highly pipelineable, which lends the QRNS its ability to perform high-throughput complex arithmetic.

4.2.3 Output Conversion

The purpose of output conversion is to recover a complex integer from its complete set of QRNS residues. QRNS output conversion first requires that the CRNS residues are obtained from the QRNS pairs. Symbolically, we are applying the mapping ϕ_M^{-1}, which is a composition of the mappings $\phi_{p_i}^{-1}$ to the collection of QRNS pairs. Recall

from Eq. 4.2 that the CRNS residues are recovered from a QRNS pair according to

$$\phi_{p_i}^{-1}\left((z_i, z_i^*)\right) = \left(2_i^{-1}(z_i + z_i^*) \bmod p_i\right) + j\left((2_i\hat{j}_i)^{-1}(z_i - z_i^*) \bmod p_i\right).$$

Suppose that the collection of QRNS pairs correspond to the complex integer $X + jY$. What the mapping $\phi_{p_i}^{-1}$ does is performing the conversions from the QRNS to the CRNS according to

$$\phi_{p_i}^{-1}\left((z_i, z_i^*)\right) = x_i + jy_i$$

where the x_i and the y_i are the real and imaginary parts of $X + jY$, respectively. The CRT can then be applied to the collection of x_i and the collection of y_i separately.

More practically, we can apply our scaling CRT and incorporate the QRNS to CRNS conversions directly into the tables. There will be two scaling CRTs performed simultaneously: one for the real and one for the imaginary part. The tables in the real CRT can be preceded by an adder to produce the sum $(z_i + z_i^*)$. The real tables now perform the function

$$\alpha_i = [[m_i\left(n_i 2_i^{-1}(z_i + z_i^*) \bmod p_i\right)/d]].$$

The tables in the imaginary CRT can be preceded by a subtractor to produce $(z_i - z_i^*)$ and the imaginary tables perform the function

$$\gamma_i = [[m_i\left(n_i(2_i\hat{j}_i)^{-1}(z_i + z_i^*) \bmod p_i\right)/d]].$$

In this case, the output of the complex scaled CRT is $X_s + jY_s$, where

$$X_s = \left(\sum_{i=1}^{L} \alpha_i \bmod 2^k\right) \bmod 2^k$$

and

$$Y_s = \left(\sum_{i=1}^{L} \gamma_i \bmod 2^k\right) \bmod 2^k.$$

The parameters d and k are as described in the section on the scaled CRT. A scaling CRT for the QRNS is shown in Fig. 4.3

Recall that the signed QRNS maps positive to the lower half of the dynamic range and negative numbers to the upper half. As with the RNS, this usually represents a difficulty in interfacing with conventional systems since the magnitude of a negative number is given by $M - |x|$. Again, the scaled QRNS CRT uses a reduced dynamic range which is a power of 2. Thus, the negative numbers are given by $2^n - |x|$, which is the standard two's-complement form for a negative number x.

4.2.4 Logarithmic Finite Field Addition

Through the use of the Chinese remainder theorem and number theoretic logarithms, a complex multiply operation is reduced to just two real addditions in the QRNS. This improvement, however, comes at the expense of an increase in the complexity of addition. In order to facilitate multiplier-less complex arithmetic, the internal numeric representation is logarithmic. A multiplication operation is performed by logarithm addition. When an addition operation is performed, the summands must be converted from logarithmic form to their ordinary representations in the finite field. The sum is then converted back to logarithmic form. In this paper, a technique is presented that eliminates the need for internal data conversions.

In typical situations, one of the operands is a constant and is known *a priori*. The other operand is produced through some pre-processing as in the case of the QRNS forward mapping. Thus, it is reasonable to assume that one of the operands is already in logarithm form and that the other operand can be delivered in exponent form by the pre-processing system. If it is desired to maintain the logarithmic data format

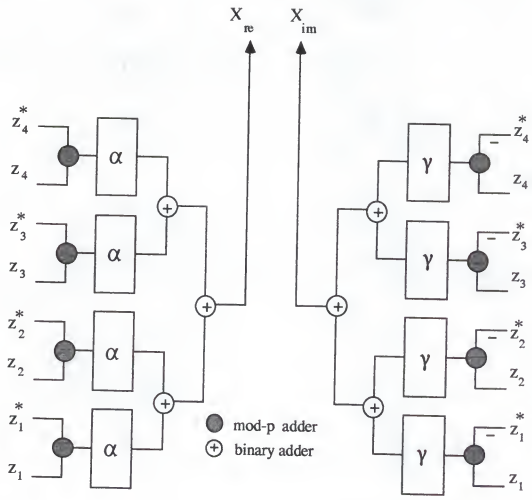


Figure 4.3. Scaling QRNS CRT engine

internally, the finite field multiplier consists of nothing more than a modulo-$(p-1)$ adder.

Suppose now that the finite field ALU uses logarithms as its internal data representation. To perform a Z_p addition, the logarithms e_x and e_y must first be converted to x and y, added modulo p, and the sum returned to logarithm form. Typically, this involves one modulo-p adder and three tables (*i.e.*, one table for each of the data format conversions).

Instead, suppose the sum of x and y is $z = \beta^{e_z}$. Write the sum as

$$\beta^{e_z} = \beta^{e_x} + \beta^{e_y} = \beta^{e_x}(1 + \beta^{e_y - e_x}).$$

Taking Z_p-logarithms of both sides,

$$e_z = \log_\beta \beta^{e_x} + \log_\beta(1 + \beta^{e_y - e_x}) = e_x + \log_\beta(1 + \beta^{e_y - e_x}).$$

If the term $\log_\beta(1 + \beta^{e_y - e_x})$ is denoted by $f(e_y - e_x)$, we have

$$e_z = e_x + f(e_y - e_x).$$

This is easily realized by subtracting e_x from e_y, implementing the function f as a table look-up, and adding e_x to the result. This requires one table and two modulo-$(p-1)$ adders. Since a modulo-p adder is considerably smaller than a small table, this represents a substantial reduction in VLSI area.

Because the element 0 does not have a logarithm over Z_p, we call the logarithm of zero "e_{NAN}" (for "not a number"). The only potential difficulty occurs when either of the logarithms is e_{NAN} or if the sum will generate e_{NaN} for the logarithm of z. Clearly, if $e_x = e_{NaN}$, then $e_z = e_y$. If $e_y = e_{NaN}$, then $e_z = e_x$. If both $e_x = e_{NaN}$ and $e_y = e_{NaN}$, then $e_z = e_{NaN}$. What if, however, $x + y = 0 \bmod p$ but neither x nor y is zero?

Theorem 4.3. $1 + \beta^{e_y - e_z} = 0 \bmod p$ *if and only if* $x + y = 0 \bmod p$. *That is to say, the only time the table needs to set the resultant logarithm to* e_{NaN} *is when* $x + y = 0 \bmod p$.

Proof: Suppose $x + y = 0 \bmod p$. Then $x = -y \bmod p$ so that

$$x^{-1} = -y^{-1} \bmod p.$$

Then

$$1 + \beta^{e_y - e_z} = 1 + \beta^{e_y}(\beta^{e_z})^{-1}.$$

But

$$1 + \beta^{e_y}(\beta^{e_z})^{-1} = 1 + x^{-1}y = 1 - y^{-1}y = 0.$$

Thus, $1 + \beta^{e_y - e_z} = 0 \bmod p$.

Next, suppose $1 + \beta^{e_y - e_z} = 0 \bmod p$. The following argument shows that $e_y - e_x$ must be equal to $(p-1)/2$. Let $Q = (p-1)/2$. Then

$$(\beta^Q)^2 = \beta^{p-1} = 1 \bmod p$$

so that either $\beta^Q = 1$ or $\beta^Q = -1$. The former is impossible since β is a primitive element of order $(p-1)$. Thus, $\beta^Q = -1 \bmod p$. But $e_y - e_x$ is surely between 0 and $(p-1)$ so that

$$\beta^{e_y - e_z} = \beta^{(p-1)/2}$$

guarantees that $e_y - e_x = (p-1)/2$. Thus,

$$e_y = e_x + (p-1)/2$$

which gives

$$x + y = \beta^{e_z} + \beta^{e_y} = \beta^{e_z} + \beta^{(p-1)/2}\beta^{e_z}.$$

But $\beta^{(p-1)/2} = -1 \bmod p$ so

$$x + y = \beta^{e_x} - \beta^{e_x} = 0 \bmod p$$

and we are done. □

The previous theorem is important because it says that if the difference between the logarithms is equal to $(p-1)/2$, the table contents can be programmed to set the output of the adder to zero and the NaN tag to 1. Besides the cases where either or both of the logarithms are NaN, there are no other pathologies about which to be concerned. The following table summarizes the possible cases that the logarithms and NaN flags can assume.

e_x	NaN_x	e_y	NaN_y	e_z	NaN_z
e_{NaN}	1	e_y	0	e_y	0
e_x	0	e_{NaN}	1	e_x	0
e_{NaN}	1	e_{NaN}	1	e_{NaN}	1
e_x	0	$e_x + (p-1)/2$	0	e_{NaN}	1
e_x	0	e_y	0	$e_x + \log_\beta(1 + \beta^{e_y - e_x})$	0

The design of a logarithm-based Z_p-adder is shown in Fig. 4.4 as compared to the earlier design which required data format conversion. Notice that one of the ROMs has been eliminated at the expense of introducing some extra combinational logic. The combinational logic is used to set e_z to its proper value and produce the proper NaN tag for e_z in accordance with the above table. This combinational logic consists of simple gates and multiplexers and is substantially smaller than a ROM.

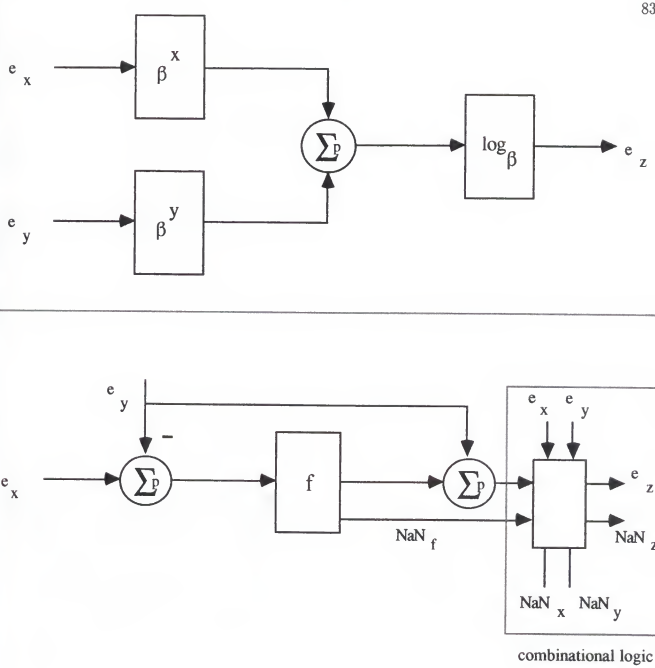


Figure 4.4. Current logarithmic Z_p-adder (top) and reported logarithmic Z_p- adder (bottom)

CHAPTER 5
THE RNS AND DIGITAL FILTERING

5.1 Digital Filtering

The two principal types of digital filters we will consider are infinite impulse response (IIR) and finite impulse response (FIR).

An IIR filter is a recursive system whose input/output behavior is described by the difference equation

$$y_k = \sum_{i=0}^{N} b_i u_{k-i} - \sum_{i=1}^{M} a_i y_{k-i}$$

where $M \leq N$. The $\mathcal{Z}$-transform of the above difference equation leads to the IIR transfer function

$$h(z) = \frac{y(z)}{u(z)} = \frac{\sum_{i=0}^{M} a_i z^{-i}}{1 + \sum_{i=1}^{N} b_i z^{-i}}.$$

An FIR is a non-recursive system whose input/output behavior is described by the difference equation

$$y_k = \sum_{i=0}^{M} b_i u_{k-i}.$$

The $\mathcal{Z}$-transform of the above difference equation leads to the FIR transfer function

$$h(z) = \frac{y(z)}{u(z)} = \sum_{i=0}^{M} b_i z^{-i}.$$

IIR filters are characterized by high frequency selectivity. That is, in order to achieve a steep magnitude frequency characteristic, the order of the IIR (namely,

M and N) is much smaller than the order required of an FIR to meet the same specifications [53]. The IIR, however, introduces a large amount of phase distortion whereas the FIR is capable of achieving a linear phase profile. Thus, the FIR is generally the desirable choice when linear phase is required.

It is also known that IIR filters are affected much more adversely than FIR filters by finite wordlength effects. That is, if a finite precision system is used to represent data and coefficients, these imprecisions are recirculated through the IIR due to its recursive nature. This can manifest itself in instabilities and limit cycling and large-amplitude distortions in magintude profile. The fixed-point behavior of an FIR is much more well-behaved and simple to analyze

The principal limitation of FIR filters is the high order required in order to meet a desired magnitude response. Filter order translates directly into multiply/accumulate cycles per filter output. As an example, in order to meet a small transition band requirement, an elliptic IIR filter of order as small as six may suffice whereas an FIR may need to be of order as great as 256. In order to produce one output sample, the IIR requires only 12 multiply/accumulates, while the FIR requires 256. Thus, if the same multiply/accumulate engine were used for both filters, the IIR could operate at a peak bandwidth in excess of 12 times greater than the FIR.

As mentioned before, however, linear phase and well-tempered fixed-point behavior render the FIR a desirable choice in a high-performance filtering environment. Thus, any method that can increase the peak bandwidth of an FIR filter should be welcome. The FIR is characterized by long strings of multiply/accumulate operations with no division, sign detection, or magnitude comparison. It is precisely the

multiply/accumulate-intensive nature of the FIR that makes it an ideal candidate for RNS implementation.

5.2 The RNS FIR

The RNS FIR is shown in Fig. 5.1. Notice that data needs to be converted to residue format, where the FIR computation can be performed in each of the modulus channels simultaneously. Upon completion of the parallel FIR cycles, the output data needs to be converted back to integer format.

Because of the small size of the RNS multiply/accumulate units, several MACs can be placed in a semi-systolic linear array, as shown in Fig. 5.1. The input data is broadcast to all of the cells simultaneously, while the tap weights propagate through the unit delay elements. It is easy to see that after initial latency, a new output sample is available each clock cycle at successive accumulator outputs.

The only significant issue that needs to be discussed in order to analyze the RNS FIR is dynamic range. Unlike conventional fixed-point systems, the RNS loses significance *entirely* if the modulus is overflowed. Thus, we need to establish dynamic range bounds by studying a worst case scenario.

Assume that we have a signed RNS, the product of whose moduli equals M. Then the dynamic range is equal to $M/2$. That is, if an accumulated result is greater than $M/2$, the result is unrecoverable. Assume that the order of the FIR is N and that data (u_k) and coefficients (b_k) are represented as two's complement integers with $B_d + 1$ and $B_c + 1$ bits, respectively. Recall that

$$y_k = \sum_{i=0}^{N} b_i u_{k-i}.$$

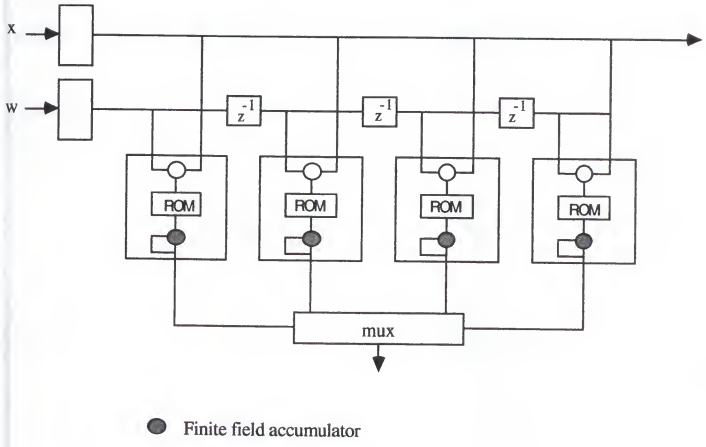


Figure 5.1. The RNS FIR

But

$$|y_k| \leq \left| \sum_{i=0}^{N} b_i u_{k-i} \right|,$$

and by the Cauchy-Schwartz inequality,

$$|y_k| \leq \sum_{i=0}^{N} |b_i||u_{k-i}|.$$

Because data and coefficients are limited to B_d and B_c bits, we have

$$|b_i| < 2^{B_d}$$

and

$$|u_{k-i}| < 2^{B_c}$$

so that

$$|y_k| < \sum_{i=0}^{N} 2^{B_d+B_c} = (N+1)2^{B_c+B_d}.$$

To insure that dynamic range overflow does not occur, we require that

$$M/2 > (N+1)2^{B_c+B_d}.$$

If W is the required wordwidth of our dynamic range,

$$W \geq \lceil \log_2(N+1) \rceil + B_c + B_d + 1. \tag{5.1}$$

For example, suppose we wish to implement an FIR of length 255 where the data and coefficients are 12-bit two's complement. Then Eq. 5.1 says that the dynamic range must be at least 33 bits in size. This is easily covered by a five modulus system where each modulus is seven bits in width.

A seven bit modulus allows the design of a very fast and compact multiplier-less multiply/accumulate unit, which is the fundamental building block of any FIR

structure. Such a unit is depicted in Fig. 5.2. This device is a finite field multiply/accumulator which is based entirely on logarithm addition. The device was designed in 2.0 micron CMOS and occupies barely 4 mm^2. Future designs will be based on a 1.0 micron technology which should yield roughly a four-fold reduction in size.

Assuming only one multiply/accumulator for each modulus, a 256-point filter takes $256 \cdot t_{MAC}$. Assuming $t_{MAC} = 15$ns, a filter cycle can be completed in about 3.8 μs, which allows real-time filtering to be performed on a data stream with a 260 kHz data rate. Multiple MACs would result in a corresponding linear increase in throughput.

As already mentioned, the small size of the finite field MAC unit allows many such devices to be placed on a single chip. Fig. 5.3 shows the layout of a chip containing 25 of these finite field MACs. These MACs could be configured as a linear array of 25 MACs or as a collection of five arrays of 5 MACs.

Consider first the case where all 25 MACs are dedicated to a single modulus. In this case, the computation of a single 256-point FIR filter cycle would take approximately 10 clock cycles. Assuming again a clock of 15ns, a complete filter cycle can be completed in roughly 150ns. This translates to a real-time data rate of 6.5 MHz. The hardware requirements of such a system, assuming five moduli for a 33-bit system, would be five chips of 25 MACs each, an array input conversion chip (which will be discussed in Chapter 11), and an array scaling CRT chip (also discussed in Chapter 11).

Consider now the second case where we configure the chip as five parallel arrays of five processing elements each. At a 15ns clock, the computation of a 256-tap

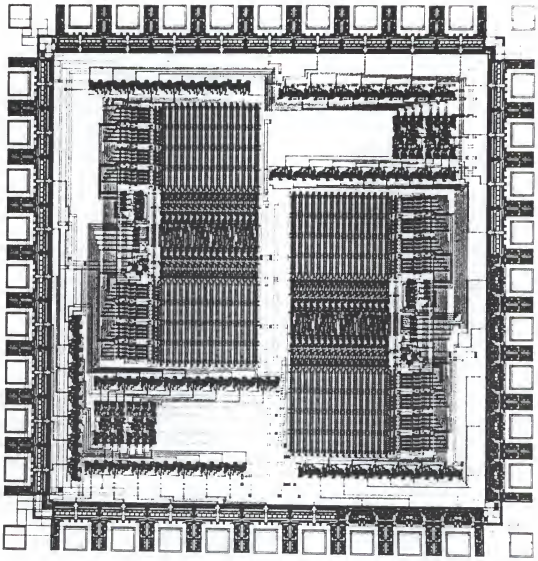


Figure 5.2. Multiplierless multiply/accumulate unit

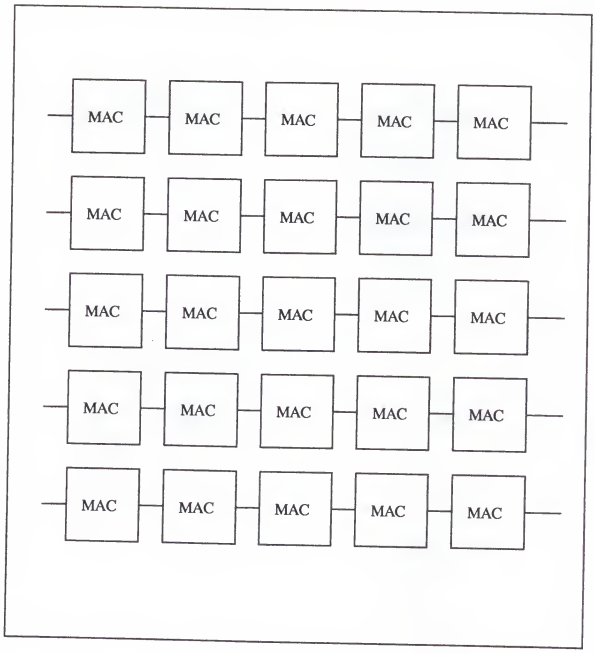


Figure 5.3. VLSI floorplan for FIR array

FIR filter cycle takes roughly 50 clock cycles, or about 750 ns. This translates to a real-time data rate of 1.3 MHz. Again, assuming a dynamic range of 33-bits and requiring five moduli, the hardware requirements include an array input conversion chip, a single FIR array chip, and an array scaling CRT chip.

5.3 An RNS Adaptive Transversal Filter

The transversal filter structure has found wide use in adaptive filtering applications such as linear prediction, echo cancelation, and interference cancelation. There are many adaptive algorithms which can be used to adapt the tap weights of the transversal filter. One of the more popular is the least mean squares (LMS) algorithm [83, 48]. This is due to its computational speed and ease of implementation.

The LMS algorithm is summarized as follows. Let $\{x_k\}$ be a discrete-time stochastic process and $\{d_k\}$ a desired response. Assume the tap weight vector is given by

$$W_k = \begin{pmatrix} w_{k1} \\ w_{k2} \\ \vdots \\ w_{kM} \end{pmatrix}.$$

Denote by X_k the input vector

$$X_k = \begin{pmatrix} x_{k1} \\ x_{k2} \\ \vdots \\ x_{kM} \end{pmatrix}.$$

Then the output of the transversal, y_k, is given by

$$y_k = W_k^T X_k.$$

The performance criterion we wish to minimize is the expectation of the mean-squared error between the output of the transversal filter and the desired response, *i.e.*,

$$\min_W(E[\epsilon^2]) = \min_W(E[(d_k - W_k^T X_k)^2]).$$

It can be shown that minimization is achieved when

$$W = R^{-1}p$$

where $R = E[X_k X_k^T]$ and $p = E[d_k X_k]$.

The LMS adaptation rule is given by

$$W_{k+1} = W_k + \mu \epsilon_k X_k.$$

Under certain assumptions [83], and for proper choice of μ, the weight vector W_k converges to the optimum value. The operation of the LMS algorithm is simple. It consists of an inner product operation followed by a set of tap-weight updates. We already know how to perform the inner product, so we need only discuss how to perform the updating.

The equation for the update of the i-th tap weight is

$$w_{k+1,i} = w_{k,i} + \mu \epsilon_k x_k.$$

We can use finite field MAC units to build a highly efficient tap-weight update element. It is possible to use logarithms alone in this cell. Specifically, the product $\mu \epsilon_k x_k$ can be generated with a pair of adders and its logarithm is then added to the logarithm of $w_{k,i}$. Such a cell is shown in Fig. 5.4. Because of its extremely small size, an array of LMS update cells can be placed in parallel to perform all of the tap-weight updates simultaneously.

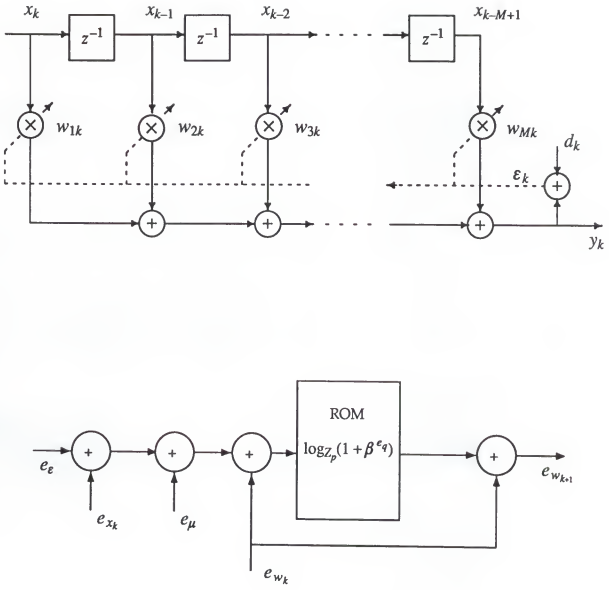


Figure 5.4. Finite field LMS tap-weight update cell

CHAPTER 6
THE POLYNOMIAL RESIDUE NUMBER SYSTEM

6.1 Introduction

Cyclic convolution of two sequences $\{f_k\}$ and $\{g_k\}$ of length N can be represented by the statement

$$q(x) = f(x)g(x) \bmod (x^N - 1).$$

Similarly, cyclic correlation can be represented by

$$p(x) = f(x)g(x) \bmod (x^N + 1).$$

Cyclic convolution and correlation are important in their own right as well as yielding simplified methods for computing linear convolution and correlation. The modular polynomial operations form the basis of the PRNS.

For the present, we will discuss the problem of computation modulo $x^N - 1$. Let F be a ring, and $F[x]$ the ring of polynomials with coefficients in F. Assume that F is such that $x^N - 1$ splits into N distinct linear factors over F, *i.e.*,

$$x^N - 1 = (x - r_0)(x - r_1)\cdots(x - r_{N-1}).$$

Then the principal ideals $(x - r_i)$ are pairwise relatively prime and the Chinese remainder theorem (CRT) [33] says that

$$F[x]/(x^N - 1) \cong F[x]/(x - r_0) \oplus \cdots \oplus F[x]/(x - r_{N-1}).$$

But $F[x]/(x - r_i) \cong F$ so that

$$F[x]/(x^N - 1) \cong F \oplus F \oplus F \oplus \cdots \oplus F := F^N. \tag{6.1}$$

Thus, the multiplication of two polynomials of degree $N-1$ requires just N multiplications in the ring F.

Equation 6.1 is the fundamental isomorphism of the PRNS. Note that the ring F has not yet been specified. Earlier work (see [60]) used the ring Z_M for certain choices of M for the case of real cyclic convolutions. When the sequences are complex, the quadratic RNS (or QRNS) [42] can be used. For the case of real sequences, however, the following theorem [31] will prove useful.

Theorem 6.1. Let p be prime. Then the congruence $x^N - 1 \equiv 0 \bmod p$ has a factorization into distinct linear factors if and only if N divides $p - 1$.

The situation when working with polynomials modulo $x^N + 1$ is similar. The condition which must be satisfied by the modulus p is as follows [31].

Theorem 6.2. Let p be prime. Then the congruence $x^N + 1 \equiv 0 \bmod p$ has a factorization into distinct linear factors if and only if N divides $(p-1)/2$.

There are certain choices of moduli which admit simple factorizations. In particular, the roots of the congruence for these moduli lead to forward and inverse PRNS mappings which are entirely shift and shift-then-add realizable. A brief example [60] is provided as a sidenote.

Corollary 6.1. If N is an even integer and $m = 2^k + 1$ with $k = Nk_1$ where k_1 is an integer, then the N roots of the congruence $x^N + 1 = 0 \bmod m$ are given by $r_t = \pm 2^{tk_1}$, $t = 1, 3, 5, \ldots, (N-1)$.

Although this leads to a simple scheme for the PRNS mappings, we regard these particular examples as "pathological"; that is, they are few and far between. We believe that it is not sound engineering practice to design RNS or PRNS hardware based on these atypical examples. Rather, our approach is to use a collection of standard cells which can be used in *every* case. This leads to hardware which is more easily designed, tested, and is less expensive. This is particularly important when dealing with a technology which is regarded as exotic and has not yet achieved mainstream acceptance.

We are now in a position to discuss the implementation of the PRNS forward and inverse mappings. Since the mappings are structurally identical for the $x^N - 1$ and the $x^N + 1$ cases, we will only consider the $x^N - 1$ case.

6.2 PRNS Forward and Inverse Mappings

If p is chosen such that $N|(p-1)$, we have the isomorphism

$$Z_p[x]/(x^N - 1) \cong Z_p^N.$$

The PRNS forward and inverse mappings are simply evaluation and interpolation, respectively. To construct the forward and inverse mappings from $Z_p[x]/(x^N - 1)$ to Z_p^N, the procedure is as follows. Let $f(x) = f_0 + f_1 x + \cdots + f_{N-1}x^{N-1}$ be a polynomial in the ring $Z_p[x]/(x^N - 1)$. The forward map is simply the canonical homomorphism

$$f(x) \rightarrow (f(r_0), f(r_1), \ldots, f(r_{N-1}))^T := \vec{\phi}(f(x)) \in Z_p^N.$$

where the r_i are the N distinct roots of $x^N - 1 = 0 \bmod p$ and can be represented in matrix form as

$$\begin{pmatrix} f(r_0) \\ f(r_1) \\ \vdots \\ f(r_{N-1}) \end{pmatrix} = \begin{pmatrix} 1 & r_0 & \cdots & r_0^{N-1} \\ 1 & r_1 & \cdots & r_1^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & r_{N-1} & \cdots & r_{N-1}^{N-1} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{pmatrix} = R\vec{f}. \tag{6.2}$$

If the matrix R is invertible and a polynomial $f(x)$ can be recovered from an N-tuple $\vec{\phi} \in Z_p^N$ by

$$\vec{f} = R^{-1}\vec{\phi}. \tag{6.3}$$

The following proves the invertibility of R. For $k = 0, 1, \ldots, N-1$, define the polynomials

$$L_k(x) = \left(\prod_{j \neq k}(x - r_j)\right)\left(\prod_{j \neq k}(r_k - r_j)\right)^{-1} = L_{k,0} + L_{k,1}x + \cdots + L_{k,N-1}x^{N-1},$$

which are the Lagrange interpolation polynomials over the field $GF(p)$. It immediately follows that $L_k(r_j) = \delta_{kj}$ (where $\delta_{kj} = 0$ if $k \neq j$ and $\delta_{kj} = 1$ if $k = j$), so the inverse of R is given by

$$R^{-1} = \begin{pmatrix} L_{0,0} & L_{1,0} & \cdots & L_{N-1,0} \\ L_{0,1} & L_{1,1} & \cdots & L_{N-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ L_{0,N-1} & L_{1,N-1} & \cdots & L_{N-1,N-1} \end{pmatrix}.$$

Equations 6.2 and 6.3 are referred to as the *PRNS forward and inverse mappings*, respectively. These mappings are computationally intensive when performed serially. Fortunately, however, the matrix R is LU-decomposable [23]. LU-decomposition of R leads to an elegant semi- systolic architecture for both the forward and inverse PRNS mappings. The following explains the LU-decomposition of R.

A matrix is LU-decomposable if all of its principal minors are nonsingular [55]. The Vandermonde structure of R guarantees that all of its principal minors are

nonsingular when the r_i are distinct so that R is LU-decomposable. To be precise, we seek a lower triangular matrix L and an upper triangular matrix U such that $R = LU$.

Define a matrix U^{-1} whose column vectors are denoted by

$$U^{-1} = (\vec{q}_0, \vec{q}_1, \cdots, \vec{q}_{N-1}).$$

Let the k-th column vector $\vec{q}_k$ correspond to some polynomial

$$\vec{q}_k \sim q_k(x) := q_{k,0} + q_{k,1}x + \cdots + q_{k,N-1}x^{N-1}.$$

By Eq. 6.2, multiplication of the k-th column vector $\vec{q}_k$ by the matrix R is the matrix representation of the mapping $\vec{\phi}(q_k(x))$, *i.e.*,

$$R\vec{q}_k = (q_k(r_0), q_k(r_1), \ldots, q_k(r_{N-1}))^T. \tag{6.4}$$

Also,

$$UU^{-1} = I = (U\vec{q}_0, U\vec{q}_1, \cdots, U\vec{q}_{N-1}),$$

so that $U\vec{q}_k = \vec{e}_k$, the k-th standard basis vector.

Now, $R\vec{q}_k = LU\vec{q}_k = L\vec{e}_k$, so the k-th column of L is given by Eq. 6.4. Hence,

$$L = \begin{pmatrix} q_0(r_0) & q_1(r_0) & \cdots & q_{N-1}(r_0) \\ q_0(r_1) & q_1(r_1) & \cdots & q_{N-1}(r_1) \\ \vdots & \vdots & \ddots & \vdots \\ q_0(r_{N-1}) & q_1(r_{N-1}) & \cdots & q_{N-1}(r_{N-1}) \end{pmatrix}, \quad \text{and} \quad U = (\vec{q}_0, \vec{q}_1, \cdots, \vec{q}_{N-1})^{-1}. \tag{6.5}$$

If U^{-1} is upper triangular, then so is U. Hence, assume that U^{-1} needs to be upper triangular. To force U^{-1} to be upper triangular, it is necessary that the polynomial $q_k(x)$ represented by the k-th column of U^{-1} be of degree $k-1$. To force L to be lower

triangular, it is necessary that $q_k(r_j) = 0$ when $k > j$. A set of $\{q_k(x)\}$ satisfying these requirements is

$$\begin{aligned} q_0(x) &= 1 \\ q_1(x) &= (x - r_0) \\ q_2(x) &= (x - r_0)(x - r_1) \\ &\cdots \\ q_{N-1}(x) &= (x - r_0)(x - r_1)\cdots(x - r_{N-2}). \end{aligned} \tag{6.6}$$

For the PRNS forward mapping, it will be required to have the diagonal elements of U^{-1} all equal to one. This can be accomplished by first finding a pair $\bar{L}, \bar{U}$ such that $\bar{L}\bar{U} = R$, and post-multiplying $\bar{U}^{-1}$ and $\bar{L}$ by a diagonal matrix D whose elements are the inverses of the diagonal entries of $\bar{U}^{-1}$.

To illustrate the utility of the LU-decomposition, let R be a 4-by-4 matrix. Assuming $x^4 - 1 = 0$ has four distinct roots, the forward mapping will send a polynomial $f(x) \in Z_p[x]/(x^4 - 1)$ to the vector $\vec{\varphi} := (\varphi_0, \varphi_1, \varphi_2, \varphi_3)^T \in F^4$. The forward mapping is represented by the equation

$$\vec{\varphi} = R\vec{f} = LU\vec{f}.$$

Define the intermediate vector $\vec{y} = U\vec{f}$. Then $\vec{y}$ can be obtained from the equation $U^{-1}\vec{y} = \vec{f}$. More explicitly,

$$\begin{pmatrix} 1 & u_{01} & u_{02} & u_{03} \\ 0 & 1 & u_{12} & u_{13} \\ 0 & 0 & 1 & u_{23} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{pmatrix},$$

which leads to the system of equations for $\vec{y}$ (back-substitution)

$$\begin{aligned} y_3 &= f_3 \\ y_2 &= f_2 - u_{23}y_3 \\ y_1 &= f_1 - u_{12}y_2 - u_{13}y_3 \\ y_0 &= f_0 - u_{01}y_1 - u_{02}y_2 - u_{03}y_3. \end{aligned} \tag{6.7}$$

Next, $\vec{\varphi} = LU\vec{f} = L\vec{y}$, *i.e.*,

$$\begin{pmatrix} \varphi_0 \\ \varphi_1 \\ \varphi_2 \\ \varphi_3 \end{pmatrix} = \begin{pmatrix} l_{00} & 0 & 0 & 0 \\ l_{10} & l_{11} & 0 & 0 \\ l_{20} & l_{21} & l_{22} & 0 \\ l_{30} & l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix},$$

which leads to the system of equations (forward substitution)

$$\begin{aligned} \varphi_0 &= l_{00}y_0 \\ \varphi_1 &= l_{10}y_0 + l_{11}y_1 \\ \varphi_2 &= l_{20}y_0 + l_{21}y_1 + l_{22}y_2 \\ \varphi_3 &= l_{30}y_0 + l_{31}y_1 + l_{32}y_2 + l_{33}y_3. \end{aligned} \tag{6.8}$$

A semi-systolic architecture for realizing Eqs. 6.7 and 6.8 is shown in Fig. 6.1.

The PRNS inverse mapping Eq. 6.3 recovers a polynomial $f(x) \in Z_p[x]/(x^N - 1)$ from the vector $\vec{\varphi} \in Z_p^N$. Since $\vec{\varphi} = LU\vec{f}$, it follows that $\vec{f} = U^{-1}L^{-1}\vec{\varphi}$. This time, it will be required that the diagonal elements of L be all ones (this can be performed in a manner similar to the forward mapping).

Since $\vec{f} = U^{-1}L^{-1}\vec{\varphi}$, define the intermediate quantity $\vec{z} = L^{-1}\vec{\varphi}$. Then $L\vec{z} = \vec{\varphi}$. Back-substitution on the above system leads to

$$\begin{aligned} z_0 &= \varphi_0 \\ z_1 &= \varphi_1 - l_{10}z_0 \\ z_2 &= \varphi_2 - l_{20}z_0 - l_{21}z_1 \\ z_3 &= \varphi_3 - l_{30}z_0 - l_{31}z_1 - l_{32}z_2. \end{aligned} \tag{6.9}$$

Next, use $\vec{f} = U^{-1}L^{-1}\vec{\varphi} = U^{-1}\vec{z}$, which yields

$$\begin{aligned} f_3 &= u_{33}z_3 \\ f_2 &= u_{22}z_2 + u_{23}z_3 \\ f_1 &= u_{11}z_1 + u_{12}z_2 + u_{13}z_3 \\ f_0 &= u_{00}z_0 + u_{01}z_1 + u_{02}z_2 + u_{03}z_3 \end{aligned} \tag{6.10}$$

A semi-systolic architecture that realizes Eqs. 6.9 and 6.10 is also shown in Fig. 6.4. Notice that the architectures of the PRNS forward and inverse mapping arrays are structurally identical. The multiplications can be performed in a multiplier-free fashion using logarithmic QRNS multipliers so that each computational element

consists of nothing more than adders and tables. The PRNS forward and inverse mapping units are constructed entirely from adders and small tables and have a structure which is easily implemented in VLSI.

6.3 The Ring $((Z_p)^2)[x]$

It is sometimes desired to embed the ring of polynomials with complex coefficients, $C[x]$, in some ring $R[x]$ in which $x^N - 1$ splits into distinct linear factors over R. This will allow us to work with complex sequences. A useful choice of the base ring R is $R = (Z_p)^2$ for some QRNS prime p. Since p is a QRNS prime, *i.e.*, $p = 4k + 1$, Theorem 1 says that N must divide $4k$ for the congruence $x^N - 1 \equiv 0 \bmod p$ to completely factor in the ring $(Z_p)[x]$. Theorem 1 can be trivially extended to the ring $((Z_p)^2)[x]$. It is clear that if $N|(p-1)$ then

$$x^N - (1,1) = (0,0)$$

will also have *at least* N distinct roots over $(Z_p)^2$ (*at least* is emphasized because $(Z_p)^2$ is not a field). Since $x^N - 1$ factors over Z_p as

$$x^N - 1 = (x - r_0)(x - r_1)\cdots(x - r_{N-1}),$$

a factorization of $x^N - (1,1)$ over $(Z_p)^2$ is

$$x^N - (1,1) = (x - (r_0, r_0))(x - (r_1, r_1))\cdots(x - (r_{N-1}, r_{N-1})) \tag{6.11}$$

The polynomial CRT then says that

$$((Z_p)^2)[x]/(x^N - (1,1)) \cong ((Z_p)^2)^N. \tag{6.12}$$

Eq. 6.12 is the fundamental equation for the PRNS when the base ring F is the ring $(Z_p)^2$. Notice that the real and imaginary parts remain decoupled in this scheme.

Thus, computations can be performed concurrently in both QRNS channels. This also holds for the forward and inverse PRNS mappings.

6.4 Dynamic Range Extension

We now return to the problem of computing with polynomials over the complex field. To obtain a larger dynamic range, several primes can be used to decompose the arithmetic into parallel channels. This is simply an application of the residue number system. If $M = p_1 p_2 \cdots p_L$, where each p_i is admissible (*i.e.*, p_i is a QRNS prime and $p_i|(N-1)$), then x^2+1 is reducible over $Z_M[x]$ and

$$Z_M[x]/(x^2+1) \cong Z_M^2.$$

But $Z_M \cong Z_{p_1} \oplus Z_{p_2} \oplus \cdots \oplus Z_{p_L}$ so that

$$Z_M[x]/(x^2+1) \cong (Z_{p_1})^2 \oplus (Z_{p_2})^2 \oplus \cdots \oplus (Z_{p_L})^2. \tag{6.13}$$

For the ring $\mathcal{R}[x]/(x^N-1)$ where $\mathcal{R} = Z_M[x]/(x^2+1)$, what results is the isomorphism

$$\mathcal{R}[x]/(x^N-1) \cong \bigoplus_{i=1}^{L}((Z_{p_i})^2)^N. \tag{6.14}$$

Equation 6.14 can be used to decompose the cyclic convolution of two complex sequences of length N to $2NL$ finite field multiplications.

The conditions on the dynamic range, M, are provided by the following theorem [31].

Theorem 6.3. Let M have prime factorization $M = p_1^{e_1} p_2^{e_2} \cdots p_L^{e_L}$. Then the congruence

$$x^N - 1 \equiv 0 \bmod M$$

has N distinct roots if and only if $N|(p_i-1)$, $\quad i = 1, 2, \ldots, L$.

The congruence can be solved by first solving the congruence over each ring Z_{p_i} and then combining the roots using the Chinese remainder theorem over the integers.

Example: Consider the factorization of the polynomial $x^4 - 1$ over the ring Z_M where M=43993. M has prime factors $p_1 = 29, p_2 = 37, p_3 = 41$, each of which satisfy the conditions in theorem 2. Each of these are QRNS primes and the congruences $x^4 - 1 \equiv 0 \bmod p_i$ will each have four distinct roots. The pertinent factorizations over each field are given in the following table.

Field	$x^4 - 1$
Z_{29}	$(x-1)(x-12)(x-17)(x-28)$
Z_{37}	$(x-1)(x-6)(x-31)(x-36)$
Z_{41}	$(x-1)(x-9)(x-32)(x-40)$

It is important that when M is not prime, $Z_M[x]$ is not a unique factorization domain. Hence, each permutation of the linear factors will still generate a valid factorization. One such factorization is given by

$$x^4 - 1 = (x - r_1)(x - r_2)(x - r_3)(x - r_4) \bmod 43993$$

where the RNS decompositions of the roots modulo 43993 are given by

$$\begin{aligned} r_1 &\leftrightarrow (1,1,1) \\ r_2 &\leftrightarrow (12,6,9) \\ r_3 &\leftrightarrow (17,31,32) \\ r_4 &\leftrightarrow (28,36,40). \end{aligned}$$

The roots modulo 43993 are obtained, of course, by applying the CRT to each three-tuple.

To achieve greater computational dynamic range, a second level of decomposition can be used. That is, computations can be performed in L parallel channels corresponding to primes $p_1, p_2, \ldots, p_L$. This requires PRNS forward and inverse mapping

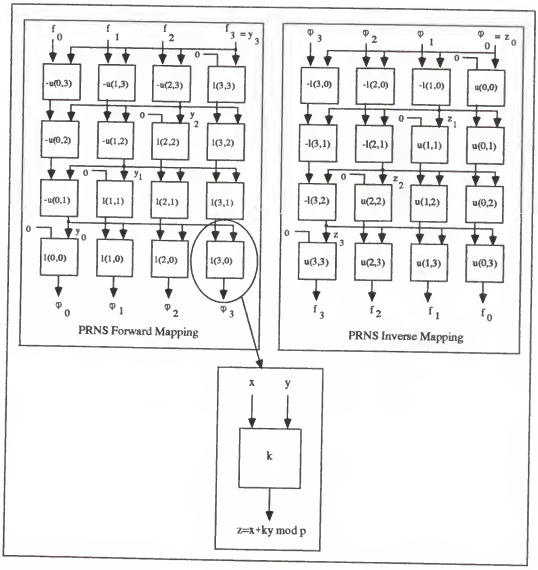


Figure 6.1. Semi-systolic arrays for PRNS forward and inverse mappings

arrays for each channel and arithmetic is performed modulo p_i in each channel. The final results are combined using the CRT, yielding a dynamic range of $M = p_1 p_2 \cdots p_L$.

6.5 The PRNS FFT

Large-blocklength DFTs can be built from highly efficient small- blocklength DFTs [15]. What needs to be developed, however, is a hardware realization for the high-speed computation of small-blocklength DFTs. From a VLSI standpoint, it is important that such a realization feature regular data flow, repetitive structure, modularity, and a good area-vs.-computational power metric. The PRNS is based on the has been shown to be a highly efficient method of performing polynomial-based computations (*e.g.*, convolution, correlation).

Both the PRNS and the RNS will be used to develop a high-speed method for the computation of small prime-blocklength DFTs. This method decomposes the computation of the DFT into two levels of modularity: algorithmic and arithmetic. The algorithmic modularity is provided by the PRNS and its ability to perform low-complexity polynomial multiplication. Arithmetic modularity is due to the RNS, in which arithmetic is performed concurrently in parallel small-wordwidth channels. An extension to the RNS, the *quadratic residue number system* (QRNS), will be used to perform complex arithmetic. The QRNS reduces a complex multiply operation from four real multiplies and two real adds to just two real additions.

The Rader prime algorithm is explained as follows. The fundamental form of a discrete Fourier transform (DFT) is

$$Y(k) = \sum_{i=0}^{p-1} W_p^{ik} y(i) \quad k = 1, \ldots, p-1 \tag{6.15}$$

A special case of this equation considers p to be prime. It is well known that the multiplicative group of Z_p is cyclic of order $p-1$, so it follows that there exists a primitive element π such that the positive powers of π generate all of the nonzero elements of Z_p. There is a bijective mapping σ from the set $\{1, 2, \ldots, p-1\}$ onto itself such that

$$\pi^{\sigma(i)} = i,$$

(*i.e.*, σ is a permutation of the indices). Therefore, Eq. 6.15 can be rewritten as

$$Y(k) = y(0) + \sum_{i=1}^{p-1} W_p^{ik} y(i), \quad k = 1, \ldots, p-1. \tag{6.16}$$

Upon substitution of the permuted indices,

$$Y(\pi^{\sigma(k)}) = y(0) + \sum_{i=1}^{p-1} W_p^{\pi^{\sigma(i)+\sigma(k)}} y(\pi^{\sigma(i)}) \tag{6.17}$$

Since σ is a permutation, letting $l = \sigma(k)$ and $j = p - 1 - \sigma(i)$ and summing over j yields

$$Y(\pi^l) = y(0) + \sum_{j=0}^{p-2} W_p^{\pi^{l-j}} y(\pi^{p-1-j}) \tag{6.18}$$

Defining the new sequences

$$Y(\pi^l) := Y(l') \quad \text{and} \quad y(j') := y(\pi^{p-1-j}),$$

Eq. 6.18 can be rewritten as

$$Y(l') = y(0) + \sum_{j=0}^{p-2} W_p^{\pi^{l-j}} y(j') \tag{6.19}$$

The sum is recognized to be a cyclic convolution of the sequences $\{W_p^{\pi^j}\}$ and $\{y(j')\}$.

For example, to compute a five-point DFT using the Rader algorithm, $W_p = e^{-j2\pi/5}$. The indices are over Z_5 and using 2 as a primitive element, use the permutation rule

$$\{2^0, 2^1, 2^2, 2^3\} = \{1, 2, 4, 3\}$$

so that the Rader prime algorithm is defined as the 4-point cyclic convolution of the sequences

$$\{W_p^{\pi^j}\} = \{W_p, W_p^2, W_p^4, W_p^3\}$$

and

$$\{y(\pi^{p-1-j})\} = \{y(1), y(3), y(4), y(2)\}.$$

to produce the sequence $\{(Y(\pi^l) - y(0))\}$ with elements

$$\{(Y(1) - y(0)), (Y(2) - y(0)), (Y(4) - y(0)), (Y(3) - y(0))\}.$$

The d.c. term $Y(0)$ is computed as the sum

$$Y(0) = \sum_{i=0}^{4} y(i)$$

and the nonzero harmonics are computed by first computing the sum in Eq. 6.19 and then adding the term $y(0)$.

In polynomial form, if

$$w(x) = W_p x^3 + W_p^4 x^2 + W_p^2 x + W_p \tag{6.20}$$

and

$$y(x) = y(2)x^3 + y(4)x^2 + y(3)x + y(1), \tag{6.21}$$

then

$$\gamma(x) = w(x)y(x) \bmod (x^4 - 1)$$

where $\gamma(x)$ is the polynomial given by

$$(Y(3) - y(0))x^3 + (Y(4) - y(0))x^2 + (Y(2) - y(0))x + (Y(1) - y(0)). \tag{6.22}$$

It would therefore seem reasonable that since the PRNS is so well-suited to computing cyclic convolutions, the production of a DFT could be made very efficient.

The PRNS FFT can be summarized as follows.

1. Given a sequence $\{y(k)\}$, evaluate the d.c. term $Y(0)$.

2. Map the polynomial $y(x)$ to the polynomial $y_Q(x)$, a polynomial with QRNS coefficients over the ring Z_p^2 which is obtained by applying Eq. 4.1 to the complex coefficients of $y(x)$.

3. Evaluate the polynomial $y_Q(x)$ at all of the roots

$$\{(r_0, r_0), (r_1, r_1), \ldots, (r_{N-1}, r_{N-1})\}$$

of $x^N - (1,1)$ over Z_p^2. This is simply an application of the PRNS forward mapping (Eq. 6.2) and results in an N- tuple of QRNS 2-tuples Φ. A pair of forward mapping arrays is needed because of the nature of the ring Z_p^2. Notice, however, that the arrays operate independently, since the QRNS eliminates the "crosstalk" between real and imaginary parts. Analogously, denote by Ω the set of *precomputed* values obtained by evaluating $w_Q(x)$ at the roots r_i.

4. Compute the product (by logarithm addition) $\Gamma = \Phi \cdot \Omega$ componentwise in each QRNS channel.

5. Recover the QRNS polynomial $\gamma_Q(x)$. This is an application of the PRNS inverse mapping (Eq. 6.3) over the ring $(Z_p)^2[x]$ and can be performed on an array pair that is identical in structure to the one used for the forward mapping in step 2.

6. Apply Eq. 4.2 to the QRNS polynomial $\gamma_Q(x)$ to obtain the polynomial $\gamma(x)$ with complex coefficients. This yields the non-zero harmonics of the DFT (offset by the value of $y(0)$, which can easily be added back in) in scrambled order.

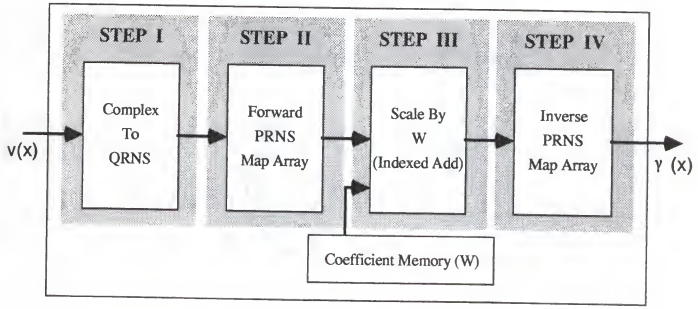


Figure 6.2. Block diagram of PRNS DFT engine

A block diagram of a single-modulus PRNS DFT engine is shown in Fig. 6.2.

We can now consider the hardware costs of a five-point DFT engine. As will be seen in Chapter 10, we have developed array input conversion and scaling CRT chips. These can be used for the data conversion required by the QRNS. The PRNS forward mapping and pointwise DFT multiplies can be combined on a single chip; the MAC array described in Chapter 5. Likewise, the PRNS inverse mapping may be performed on an identical chip.

Thus, for a three modulus system which provides roughly 24 bits of dynamic range, we require an array input conversion chip, six MAC array chips for the PRNS forward mappings and DFT multiplies for the I and Q channels, six MAC chips for the PRNS inverse mappings, and a array scaling CRT chip. After initial latency, a complete five-point DFT is delivered each clock cycle. Assuming a 15ns clock time, we can achieve five point DFTs at a sustained throughput rate of 67 MHz. From this extremely high-throughput DFT engine, we can construct longer blocklength FFTs using the Cooley-Tukey FFT Algorithm or Good-Thomas ordering.

As with all of our designs, the primitive architectural structure is again the multiply/accumulate and that the entire design is based on adders and small tables. Because of its small size, an array of PRNS DFT engines could be integrated into an array for parallel computation of FFTs.

6.6 2-D Cyclic Convolution

Let $h(n_1, n_2)$ and $f(n_1, n_2)$, $0 \leq n_1 \leq N_1 - 1$, $0 \leq n_2 \leq N_2 - 1$ be two discrete time two-dimensional signals. Then the cyclic convolution of h and f is given by

$$(\mathbf{h} \otimes \mathbf{f})_{n_1,n_2} = g(n_1, n_2) = \sum_{m_1=0}^{N_1-1} \sum_{m_2=0}^{N_2-1} h(m_1, m_2) f\left((n_1 - m_1)_{N_1}, (n_2 - m_2)_{N_2}\right) \quad (6.23)$$

where $(\cdot)_N$ indicates reduction modulo N.

There is also a polynomial representation of cyclic convolution. Associate with the sequences $h(n_1, n_2)$ and $f(n_1, n_2)$ the polynomials

$$h(x,y) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} h(n_1,n_2) x^{n_1} y^{n_2} \tag{6.24}$$

and

$$f(x,y) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} f(n_1,n_2) x^{n_1} y^{n_2}. \tag{6.25}$$

Then the cyclic convolution of h and f has the polynomial representation

$$g(x,y) = h(x,y) f(x,y) \pmod{x^{N_1}-1} \pmod{y^{N_2}-1}. \tag{6.26}$$

6.7 Chinese Remainder Theorem Over $R[x,y]$

As before, assume that $R = Z_M$ where M is chosen so that $x^{N_1}-1$ and $y^{N_2}-1$ have N_1 and N_2 distinct linear factors, respectively (necessary and sufficient conditions for this to be true will be given in a later section). In other words,

$$x^{N_1} - 1 = (x - r_0)(x - r_1) \cdots (x - r_{N_1-1}) \tag{6.27}$$

and

$$y^{N_2} - 1 = (y - s_0)(y - s_1) \cdots (y - s_{N_2-1}). \tag{6.28}$$

The Chinese Remainder Theorem (CRT) for polynomial rings says that

$$R[x,y]/(x^{N_1}-1) \cong R[x,y]/(x-r_0) \oplus R[x,y]/(x-r_1) \oplus \cdots \oplus R[x,y]/(x-r_{N_1-1}). \tag{6.29}$$

However, $R[x,y]/(x-r_i)$ is simply the ring $R[y]$ so that

$$R[x,y]/(x^{N_1}-1) \cong \underbrace{R[y] \oplus R[y] \oplus \cdots \oplus R[y]}_{N_1} := (R[y])^{N_1}.$$

The forward mapping is given by

$$f(x,y) \in R[x,y]/(x^{N_1}-1) \rightarrow (f(r_0,y), f(r_1,y), \ldots, f(r_{N_1-1},y)) \tag{6.30}$$

Now, the ring

$$\left(R[x,y]/(x^{N_1}-1)\right)/(y^{N_2}-1) = (R[y])^{N_1}/(y^{N_2}-1)$$

needs to be decomposed. Because $y^{N_2}-1$ has the factorization given by Eq. 6.28, and since $R[y]/(y-s_i) \cong R$, it can be shown that

$$\left(R[x,y]/(x^{N_1}-1)\right)/(y^{N_2}-1) \cong \left(\underbrace{R \oplus \cdots \oplus R}_{N_1}\right)^{N_2} := R^{N_1 N_2} \tag{6.31}$$

The isomorphism in Eq. 6.31 can be represented by mapping the polynomial $f(x,y)$ to the matrix of polynomial residues

$$\mathbf{F} = \begin{pmatrix} f(r_0,s_0) & f(r_0,s_1) & \cdots & f(r_0,s_{N_2-1}) \\ f(r_1,s_0) & f(r_1,s_1) & \cdots & f(r_1,s_{N_2-1}) \\ \vdots & \vdots & \vdots & \vdots \\ f(r_{N_1-1},s_0) & f(r_{N_1-1},s_1) & \cdots & f(r_{N_1-1},s_{N_2-1}) \end{pmatrix} \tag{6.32}$$

A procedure is needed to recover the polynomial $f(x,y)$ from its matrix of residues. To accomplish this, introduce the interpolation polynomials

$$L_j(y) = N_2^{-1} \sum_{k=0}^{N_2-1} s_j^{-k} y^k, \quad j = 0,1,\ldots,N_2-1 \tag{6.33}$$

and

$$T_n(x) = N_1^{-1} \sum_{m=0}^{N_1-1} r_n^{-m} x^m, \quad n = 0,1,\ldots,N_1-1. \tag{6.34}$$

In Part I, it was shown that $L_j(s_i) = \delta_{ij}$ and that $T_n(r_q) = \delta_{nq}$. It then follows that each $f(r_i,y)$ can be recovered from the set of residues $\{f(r_i,s_j) \mid j = 0,1,\ldots,N_2-1\}$ by the formula

$$f(r_i,y) = \sum_{j=0}^{N_2-1} f(r_i,s_j) L_j(y). \tag{6.35}$$

Once the $f(r_i, y)$, $i = 0, 1, \ldots, N_1 - 1$, have been constructed, $f(x, y)$ can be recovered from the set of polynomials $f(r_i, y)$ by the formula

$$f(x,y) = \sum_{n=0}^{N_1-1} f(r_n, y) T_n(x). \tag{6.36}$$

Substituting Eq. 6.34 into Eq. 6.36 and changing the order of summation yields

$$f(x,y) = N_1^{-1} \sum_{n=0}^{N_1-1} \sum_{m=0}^{N_1-1} r_n^{-m} x^m f(r_n, y), \tag{6.37}$$

and upon substituting Eq. 6.33 into Eq. 6.37,

$$f(x,y) = N_1^{-1} \sum_{n=0}^{N_1-1} \sum_{m=0}^{N_1-1} \sum_{j=0}^{N_2-1} r_n^{-m} x^m f(r_n, s_j) L_j(y),$$

which, by Eq. 6.33, can be written as

$$f(x,y) = N_1^{-1} N_2^{-1} \sum_{n=0}^{N_1-1} \sum_{m=0}^{N_1-1} \sum_{j=0}^{N_2-1} \sum_{k=0}^{N_2-1} x^m y^k r_n^{-m} s_j^{-k} f(r_n, s_j). \tag{6.38}$$

Interchanging the order of summation,

$$f(x,y) = N_1^{-1} N_2^{-1} \sum_{m=0}^{N_1-1} \sum_{k=0}^{N_2-1} x^m y^k \left(\sum_{n=0}^{N_1-1} \sum_{j=0}^{N_2-1} r_n^{-m} s_j^{-k} f(r_n, s_j) \right). \tag{6.39}$$

Finally, Eq. 6.39 can be rearranged in the following form:

$$f(x,y) = N_1^{-1} N_2^{-1} \sum_{m=0}^{N_1-1} \sum_{k=0}^{N_2-1} \beta_{mk} x^m y^k \tag{6.40}$$

where

$$\beta_{mk} := \sum_{n=0}^{N_1-1} \sum_{j=0}^{N_2-1} r_n^{-m} s_j^{-k} f(r_n, s_j). \tag{6.41}$$

Notice that Eq. 6.41 is a quadratic form with weight matrix $\mathbf{F}$. If the vectors $\mathbf{r_m}$ and $\mathbf{s_k}$ are defined by

$$\mathbf{r_m} := \begin{pmatrix} r_0^{-m} \\ r_1^{-m} \\ \vdots \\ r_{N_1-1}^{-m} \end{pmatrix} \quad \text{and} \quad \mathbf{s_k} := \begin{pmatrix} s_0^{-k} \\ s_1^{-k} \\ \vdots \\ s_{N_2-1}^{-k} \end{pmatrix},$$

then

$$\beta_{mk} = (\mathbf{r}^{\mathbf{m}})^T \mathbf{F} \mathbf{s}^{\mathbf{k}}. \tag{6.42}$$

It is clear from Eq. 6.40 that

$$f(m,k) = N_1^{-1} N_2^{-1} \beta_{mk}. \tag{6.43}$$

The importance of Eq. 6.42 may at first be overlooked. The matrix $\mathbf{F}$ is the same in the computation of every element of the result. The computation of the inverse CRT has been reduced to the calculation of a quadratic form. If the modulus m has been chosen so that all of the roots of the congruences $x^{N_1} - 1 = 0$ and $y^{N_2} - 1 = 0$ are shift realizable (*e.g.*, m a Fermat prime), then the evaluation of the matrix $\mathbf{F}$ and the vectors $\mathbf{r_m}$, $\mathbf{s_k}$ will be very simple.

To actually perform a cyclic convolution of two sequences $h(n_1, n_2)$ and $f(n_1, n_2)$ where $0 \leq n_1, \leq N_1 - 1$, $0 \leq n_2 \leq N_2 - 1$, a modulus m needs to be determined so that the congruences

$$x^{N_1} - 1 \equiv 0 \bmod m \tag{6.44}$$

and

$$y^{N_2} - 1 \equiv 0 \bmod m \tag{6.45}$$

have N_1 distinct roots $\{r_0, r_1, \ldots, r_{N_1-1}\}$ and N_2 distinct roots $\{s_0, s_1, \ldots, s_{N_2-1}\}$, respectively. Theorem 2 is useful in determining the appropriate modulus.

Now, assume that the modulus has been chosen so that both Eq. 6.44 and Eq. 6.45 have N_1 and N_2 distinct roots, respectively. Now, associate with the sequences $h(n_1, n_2)$ and $f(n_1, n_2)$ the polynomials

$$h(x,y) = \sum_{n_1=0}^{N_1-1} h(n_1, n_2) x^{n_1} y^{n_2}$$

and

$$f(x,y) = \sum_{n_1=0}^{N_1-1} f(n_1,n_2)x^{n_1}y^{n_2}.$$

Now, map $f(x,y)$ and $h(x,y)$ to the matrices $\mathbf{H}$ and $\mathbf{F}$, where

$$\mathbf{H}_{ij} = (h(r_i,s_j)) \bmod m, \quad i = 0,1,\ldots,N_1-1, \quad j = 0,1,\ldots,N_2-1$$

and

$$\mathbf{F}_{ij} = (f(r_i,s_j)) \bmod m, \quad i = 0,1,\ldots,N_1-1, \quad j = 0,1,\ldots,N_2-1.$$

Now, compute the *pointwise* matrix product

$$\mathbf{G}_{ij} = \mathbf{H}_{ij} \cdot \mathbf{F}_{ij} = (h(r_i,s_j)f(r_i,s_j)) \bmod m. \tag{6.46}$$

It then follows from Eq. 6.40 that the (n_1,n_2)- th element of the resulting sequence $g(n_1,n_2)$ is given by

$$g(n_1,n_2) = (\mathbf{r_{n_1}})^T(\mathbf{G}\mathbf{s_{n_2}}).$$

Example: Consider the sequences

$$h(n_1,n_2) = \{h(0,0),h(0,1),h(1,0),h(1,1)\} = \{1,0,2,1\} \tag{6.47}$$

and

$$f(n_1,n_2) = \{f(0,0),f(0,1),f(1,0),f(1,1)\} = \{1,0,1,1\}. \tag{6.48}$$

By, Eqs. 6.24 and 6.25, associate with the sequences the polynomials

$$h(x,y) = 1 + 2x + xy, \qquad \text{and } f(x,y) = 1 + x + xy.$$

The polynomial representation of the cyclic convolution is given by

$$g(x,y) = h(x,y)f(x,y) \pmod{x^2-1} \pmod{y^2-1}.$$

(In this case, $N_1 = N_2 = 2$). The modulus m needs to be chosen so that $2|(m-1)$. Let $m = 7$ so that the polynomials are being factored over the finite field $GF(7)$. The factorization of $x^2 - 1$ and $y^2 - 1$ over $GF(7)$ is

$$x^2 - 1 = (x-1)(x-6) \quad \text{and } y^2 - 1 = (y-1)(y-6).$$

Let $r_0 = s_0 = 1$ and $r_1 = s_1 = 6$. Then the matrices $\mathbf{F}$ and $\mathbf{H}$ are given by

$$\mathbf{F} = \begin{pmatrix} f(1,1) & f(1,6) \\ f(6,1) & f(6,6) \end{pmatrix} = \begin{pmatrix} 4 & 2 \\ 5 & 0 \end{pmatrix}$$

and

$$\mathbf{H} = \begin{pmatrix} h(1,1) & h(1,6) \\ h(6,1) & h(6,6) \end{pmatrix} = \begin{pmatrix} 3 & 1 \\ 6 & 1 \end{pmatrix}$$

and the pointwise product is

$$\mathbf{G} = \mathbf{H} \cdot \mathbf{F} = \begin{pmatrix} 5 & 2 \\ 2 & 0 \end{pmatrix}.$$

To find, for instance, the value of $g(1,1)$, use Eq. 6.43. Recognizing that $1^{-1} = 1$ and $6^{-1} = 6$,

$$g(1,1) = N_1^{-1} N_2^{-1} \beta_{11}$$

where

$$\beta_{11} = (\mathbf{r_0})^T \mathbf{G} (\mathbf{s_k}).$$

Formally,

$$\beta_{11} = \begin{pmatrix} 1 & 6 \end{pmatrix} \begin{pmatrix} 5 & 2 \\ 2 & 0 \end{pmatrix} \begin{pmatrix} 3 \\ 2 \end{pmatrix} = 1 \bmod 7.$$

Then

$$g(1,1) = 2^{-1} \cdot 2^{-1} \cdot 1 = 2,$$

which is the correct answer.

6.8 The 2-D Rader Algorithm

Let $v(i,j)$ be a two-dimensional sequence where $0 \leq i \leq p-1$ and $0 \leq j \leq q-1$ where both p and q are prime. Then the 2-D Fourier transform of $v(i,j)$ is the sequence

$$V(m,n) = \sum_{i=0}^{p-1} \sum_{j=0}^{q-1} w^{im} \mu^{jn} v(i,j),$$

where $0 \leq m \leq p-1$ and $0 \leq n \leq q-1$. Four cases need to be treated separately: Case 1) $m = n = 0$; Case 2) $n = 0$; Case 3) $m = 0$; Case 4) $m, n \neq 0$. For the present, it will suffice to consider efficient computation for Case 4. The expression for the 2-D DFT can be rewritten as,

$$V(m,n) = v(0,0) + \sum_{i=1}^{p-1} \omega^{im} v(i,0) + \sum_{j=1}^{q-1} \mu^{jn} v(0,j) + \sum_{i=1}^{p-1} \sum_{j=1}^{q-1} \omega^{im} \omega^{jn} v(i,j). \quad (6.49)$$

To expedite the discussion, we will limit our attention to the double summation. Thus, define the quantity $\hat{V}$ by

$$\hat{V}(m,n) = \sum_{i=1}^{p-1} \sum_{j=1}^{q-1} \omega^{im} \omega^{jn} v(i,j). \quad (6.50)$$

Since p and q are prime, the multiplicative groups of $GF(p)$ and $GF(q)$ are cyclic. As before, there exist permutations

$$\sigma : GF(p) - \{0\} \rightarrow GF(p) - \{0\}$$

and

$$\tau : GF(q) - \{0\} \rightarrow GF(q) - \{0\}$$

and primitive elements $\pi \in GF(p)$, $\gamma \in GF(q)$ such that both

$$\pi^{\sigma(i)} = i, \quad \pi^{\sigma(m)} = m$$

and

$$\gamma^{\tau(j)} = j, \quad \gamma^{\tau(n)} = n.$$

Using this "reindexing", Eq. 6.50 can be rewritten as

$$\hat{V}(\pi^{\sigma(m)}, \gamma^{\tau(n)}) = \sum_{i=1}^{p-1} \sum_{j=1}^{q-1} \omega^{\pi^{\sigma(i)+\sigma(m)}} \mu^{\gamma^{\tau(j)+\tau(n)}} v(\pi^{\sigma(i)}, \gamma^{\tau(j)}). \tag{6.51}$$

Now, since σ and τ are bijective, make the change of indices

$$l_1 = \sigma(m), \quad s_1 = p - 1 - \sigma(i)$$

and

$$l_2 = \tau(n), \quad s_2 = q - 1 - \tau(j).$$

Using these new indices and the fact that $\pi^{p-1} = \gamma^{q-1} = 1$,

$$\hat{V}(\pi^{l_1}, \gamma^{l_2}) = \sum_{s_1=0}^{p-2} \sum_{s_2=0}^{q-2} \omega^{\pi^{l_1-s_1}} \mu^{\gamma^{l_2-s_2}} v(\pi^{p-1-s_1}, \gamma^{q-1-s_2}). \tag{6.52}$$

Now, defining the sequences

$$\hat{V}'(l_1, l_2) := \hat{V}(\pi^{l_1}, \gamma^{l_2}), \quad v'(s_1, s_2) := v(\pi^{p-1-s_1}, \gamma^{q-1-s_2}), \tag{6.53}$$

and

$$W(s_1, s_2) = w^{\pi^{s_1}} \mu^{\gamma^{s_2}}, \tag{6.54}$$

Eq. 6.52 is immediately recognizable as the 2- dimensional cyclic convolution of the sequences W and v'. In other words,

$$\hat{V}'(l_1, l_2) = (\mathbf{W} \otimes \mathbf{v}')(l_1, l_2). \tag{6.55}$$

This is simply a reindexing of the variables in Eq. 6.52.

Example: The following is a rather tedious (but illustrative of the reindexing scheme) example. Suppose that $v(m, n)$ is a 5 by 5 signal to be Fourier transformed.

Using Eq. 6.55, the "difficult part" in the computation of the non-zero harmonics of the Fourier transform of $v(m,n)$ is given by

$$\hat{V}'(l_1,l_2) = (\mathbf{W} \otimes \mathbf{v}')(l_1,l_2)$$

where $\hat{V}', v'$, and W are as in Eq. 6.53 and Eq. 6.54. The polynomial form of this cyclic convolution is

$$\hat{V}'(x,y) = W(x,y)v'(x,y) \pmod{x^4-1} \pmod{y^4-1}$$

where

$$\begin{aligned}\hat{V}'(x,y) &= V_{1,1} + V_{1,2}y + V_{1,4}y^2 + V_{1,3}y^3 + V_{2,1}x + V_{2,2}xy + V_{2,4}xy^2 + V_{2,3}xy^3 \\ &\quad + V_{4,1}x^2 + V_{4,2}x^2y + V_{4,4}x^2y^2 + V_{4,3}x^2y^3 + V_{3,1}x^3 + V_{3,2}x^3y \\ &\quad + V_{3,4}x^3y^2 + V_{3,3}x^3y^3\end{aligned}$$

$$\begin{aligned}v'(x,y) &= v_{1,1} + v_{1,3}y + v_{1,4}y^2 + v_{1,2}y^3 + v_{3,1}x + v_{3,3}xy + v_{3,4}xy^2 + v_{3,2}xy^3 \\ &\quad + v_{4,1}x^2 + v_{4,3}x^2y + v_{4,4}x^2y^2 + v_{4,2}x^2y^3 + v_{2,1}x^3 + v_{2,3}x^3y \\ &\quad + v_{2,4}x^3y^2 + v_{2,2}x^3y^3\end{aligned}$$

and

$$\begin{aligned}W(x,y) &= w^1\mu^1 + w^1\mu^2y + w^1\mu^4y^2 + w^1\mu^3y^3 + w^2\mu^1x \\ &\quad + w^2\mu^2xy + w^2\mu^4xy^2 + w^2\mu^3xy^3 + w^4\mu^1x^2 + w^4\mu^2x^2y \\ &\quad + w^4\mu^4x^2y^2 + w^4\mu^3x^2y^3 + w^3\mu^1x^3 + w^3\mu^2x^3y + w^3\mu^4x^3y^2 \\ &\quad + w^3\mu^3x^3y^3.\end{aligned}$$

Now, using Case 4, the actual value of the m,n-th harmonic is given by

$$V(m,n) = \hat{V}(m,n) + V(m,0) + V(0,n) - v(0,0)$$

or,

$$V'(m,n) = \hat{V}'(m,n) + V'(m,0) + V'(0,n) - v(0,0)$$

where the zero harmonics can be computed in a manner similar to the one dimensional method. Additionally, the arithmetic may be performed over the ring $(GF(p) \oplus GF(p))[x,y]$.

CHAPTER 7
COMPUTATIONAL COMPLEXITY

Several fundamental signal processing operations (*e.g.*, cyclic convolution, correlation) can be cast in polynomial form. Fast algorithms for performing these operations are often based on the problem of computing the product of two polynomials f and g modulo some other polynomial p of degree N with minimal multiplicative complexity [15, 50]. The Chinese Remainder Theorem (CRT), when applied to polynomial rings, provides a particularly efficient solution to this problem.

Recall that

$$F[x]/\langle x^N \pm 1\rangle \cong F^N.$$

This is the fundamental result upon which most fast polynomial multiplication systems are premised. The forward and inverse mappings relating the two isomorphic rings will now be developed. First, let $f(x) = f_0 + f_1 x + \cdots + f_{N-1}x^{N-1}$ be a polynomial in the ring $F[x]/\langle x^N \pm 1\rangle$. The image $\vec{\phi}$ of $f(x)$ in F^N is given by the canonical homomorphism

$$\vec{\phi} = (f(r_0), f(r_1), \ldots, f(r_{N-1}))^T \in F^N.$$

This mapping is represented in matrix form as

$$\begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_N \end{pmatrix} = \begin{pmatrix} 1 & r_0 & \cdots & r_0^{N-1} \\ 1 & r_1 & \cdots & r_1^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & r_{N-1} & \cdots & r_{N-1}^{N-1} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{pmatrix} = \mathbf{V}\vec{f}.$$

The N-by-N matrix $\mathbf{V} \in M_{N\times N}(F)$ is a Vandermonde matrix and is intimately related to the problem of Lagrange Interpolation. If $\mathbf{V}$ is invertible over $M_{N\times N}(F)$, then a polynomial $f(x)$ can be recovered from its image $\vec{\phi} \in F^N$ simply by

$$\vec{f} = \mathbf{V}^{-1}\vec{\phi}.$$

We repeat the following theorem [23] establishing the invertibility of $\mathbf{V}$.

Theorem 7.1. The Vandermonde matrix $\mathbf{V}$ *is invertible over the field* F.

Proof: For $k = 0, 1, \ldots, N-1$, define the polynomials $L_k(x)$ by

$$L_{k,0} + L_{k,1}x + \cdots + L_{k,N-1}x^{N-1} = \left(\prod_{j\neq k}(x - r_j)\right)\left(\prod_{j\neq k}(r_k - r_j)\right)^{-1},$$

which are the Lagrange interpolation polynomials over the field F. It immediately follows that $L_k(r_j) = \delta_{kj}$ (where $\delta_{kj} = 0$ if $k \neq j$ and $\delta_{kj} = 1$ if $k = j$), so the inverse of $\mathbf{V}$ is given by

$$\mathbf{V}^{-1} = \begin{pmatrix} L_{0,0} & L_{1,0} & \cdots & L_{N-1,0} \\ L_{0,1} & L_{1,1} & \cdots & L_{N-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ L_{0,N-1} & L_{1,N-1} & \cdots & L_{N-1,N-1} \end{pmatrix}. \quad \square$$

The isomorphism represented by $\mathbf{V}$ allows the product of two polynomials in the ring $F[x]/\langle x^N \pm 1\rangle$ (a computation which requires $O(N^2)$ F-multiplies) to be computed in the ring F^N (with just $O(N)$ F-multiplies). An important interpretation of this reduction in multiplicative complexity is provided by viewing the computations as occurring over an associative algebra.

It is simple to impose the structure of an associative algebra on the rings $F[x]/\langle x^N \pm 1\rangle$ and F^N. Simply maintain the multiplicative and additive structures of both rings

and let the field F act on both of the rings with the scalar-vector product defined in the obvious way. It is trivial to verify that all of the axioms are satisfied and that what results is a pair of associative algebras over F.

Ignoring the multiplicative structure, the vector space (over F) $F[x]/\langle x^N \pm 1\rangle$ is of dimension N with basis

$$\mathcal{B}_1 = \{\vec{v}_0, \vec{v}_1, \ldots, \vec{v}_{N-1}\} := \{1, x, x^2, \ldots, x^{N-1}\}$$

and hence is isomorphic to the F-vector space F^N with canonical basis

$$\mathcal{B}_2 = \{\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_N\}$$

where

$$\vec{e}_i = (0, 0, \ldots, \overset{i}{1}, 0, \ldots, 0).$$

It is thus clear that the mapping represented by $\mathbf{V}$ is a vector space isomorphism.

Theorem 7.2. The mapping represented by $\mathbf{V}$ *is an algebra isomorphism of the* F*-algebras* $F[x]/\langle x^N \pm 1\rangle$ *and* F^N.

Proof: The CRT already shows that $\mathbf{V}$ is a ring isomorphism. It is simple to show that $\mathbf{V}$ is F- linear; clearly, $\mathbf{V}(\vec{f}+\vec{g}) = \mathbf{V}\vec{f}+\mathbf{V}\vec{g}$ and for any $a \in F$, $\mathbf{V}(a\vec{f}) = (a\mathbf{V})\vec{f}$. Hence, $F[x]/\langle x^N \pm 1\rangle$ and F^N are isomorphic as F-algebras. □

An element $f(x)$ of $F[x]/\langle x^N \pm 1\rangle$ can be expressed relative to the basis $\mathcal{B}_1$ as

$$f(x) = \sum_{i=0}^{N-1} f_i \vec{v}_i$$

or its image $\vec{\phi} \in F^N$ under $\mathbf{V}$ can be represented relative to the basis $\mathcal{B}_2$ as

$$\vec{\phi} = \sum_{i=1}^{N} \phi_i \vec{e}_i.$$

Notice, however, that $\vec{e}_i \cdot \vec{e}_j = \delta_{ij}\vec{e}_i$ for all $\vec{e}_i, \vec{e}_j \in \mathcal{B}_2$. This is certainly not the case for $\mathcal{B}_1$.

To compute the product of $f, g \in F[x]/\langle x^N \pm 1\rangle$, (*i.e.*, relative to the basis $\mathcal{B}_1$), $f(x)g(x)$ is represented by

$$\left(\sum_{i=0}^{N-1} f_i\vec{v}_i\right)\left(\sum_{j=0}^{N-1} g_j\vec{v}_j\right) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f_i g_j (x^{i+j} \bmod \langle x^N \pm 1\rangle), \tag{7.1}$$

which has a multiplicative complexity of $O(N^2)$. Instead, the product can be computed by mapping f, g under $\mathbf{V}$ to $\vec{\phi}, \vec{\gamma} \in F^N$ and forming the quantity

$$\left(\sum_{i=1}^{N} \phi_i\vec{e}_i\right)\left(\sum_{j=1}^{N} \gamma_j\vec{e}_j\right) = \sum_{i=1}^{N}\sum_{j=1}^{N} \phi_i\gamma_j\vec{e}_i\vec{e}_j. \tag{7.2}$$

The nature of the basis $\mathcal{B}_2$, however, allows Eq. 7.2 to be simplified to

$$\sum_{i=1}^{N}\sum_{j=1}^{N} \phi_i\gamma_j\delta_{ij}\vec{e}_i = \sum_{i=1}^{N} \phi_i\gamma_i\vec{e}_i. \tag{7.3}$$

Although the complexity of Eq. 7.2 is still $O(N^2)$, $N(N-1)$ of the multiplies are by zero, yielding an effective complexity of $O(N)$ (the non-zero multiplies are the $\phi_i\gamma_i$ in Eq. 7.3). This "decoupling" of the basis vectors in $\mathcal{B}_2$ (which is not shared by the basis $\mathcal{B}_1$) is what allows the reduced complexity. More significantly, this basis results in the *minimum* multiplicative complexity. To show this, the following theorem [15, 86] is needed.

Theorem 7.3. Let $p(x)$, a polynomial of degree N, be a product of k distinct prime polynomials. Every algorithm to compute the product $s(x) = f(x)g(x) \bmod (p(x))$ uses at least $2N - k$ multiplications.

In our case, the polynomial $p(x) = x^N \pm 1$ is assumed to split into N distinct linear factors. Thus, no algorithm can compute the product of $f(x)$ and $g(x)$ in less than

$2N - k = 2N - N = N$ multiplications. If $k < N$, there is no isomorphism between $F[x]/\langle x^N \pm 1\rangle$ and F^N and the multiplicative complexity will always be greater than N. It is precisely the factorization of $x^N \pm 1$ into distinct linear factors that allows the change of basis from $\mathcal{B}_1$ to $\mathcal{B}_2$. In turn, this new basis renders the multiplicative complexity as small as is possible.

CHAPTER 8
ALGEBRAIC INTEGER RESIDUE NUMBER SYSTEM (AIRNS)

In a "real-life" problem, incoming data must be quantized by scaling by a large real number and rounding to the nearest integer or nearest Gaussian integer. This can offset the advantages of an RNS system. It would be desirable to have a scheme which requires little or no scaling yet still retains the parallelism of the RNS. The Algebraic Integer Number System (AIRNS) is such a system.

From the theory of algebraic extensions, we know that given a field F and a polynomial $m(x) \in F[x]$ that is irreducible over F, there exists an extension field E of F and an element $\alpha \in E$ such that $m(\alpha) = 0$ [46]. When $m(x)$ is irreducible over F, the principal ideal $(m(x))$ generated by m in $F[x]$ is maximal, and hence the quotient ring $F[x]/(m(x)) := F(\alpha)$ is a field containing an isomorphic image of F and the element $\alpha = x + (m(x)) \in F[x]/(m(x))$ is such that $m(\alpha) = 0$.

What is more, the field $F(\alpha)$ is isomorphic to the image of $F[x]$ under the evaluation homomorphism $f(x) \rightarrow f(\alpha)$ and can be regarded as a vector space over F of dimension n where $n = deg(m)$ with basis

$$\{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\}.$$

This means that any element $\beta \in F(\alpha)$ can be written *uniquely* as

$$\beta = a_0 + a_1\alpha + a_2\alpha^2 + \ldots + a_{n-1}\alpha^{n-1}.$$

For example, if the field of interest F is the reals, R, and the irreducible polynomial is $m(x) = x^2 + 1$, we obtain a second degree extension that is a vector space over the reals with basis $\{1, \alpha\}$ where α is a root of $m(x)$. If α is identified with $\sqrt{-1}$, the complex field results.

Recall that in a field, a *primitive Nth root of unity* is an element ω of multiplicative order N. For example, in the complex field, $\omega_n = e^{n2\pi j/N}$ where $\gcd(n, N) = 1$ is a primitive Nth root of unity [15].

Definition 8.1. Over any field F, for each R, the polynomial

$$C_R(x) = \prod_{(i,R)=1} (x - \omega^i)$$

is called the R-th Cyclotomic Polynomial where ω is a primitive Rth root of unity in some extension field of F.

Clearly, the cyclotomic polynomials are of degree $\phi(R)$ where ϕ is Euler's totient function. Let the field F be the rationals, Q. When $R = 2^m$, $m \geq 2$, $C_R(x) = x^N + 1$ where $N = \phi(2^m) = 2^{m-1}$ and is irreducible over the rationals. An extension field $Q(\omega)$ results that can be regarded as a vector space over the rationals with basis $\{1, \omega, \ldots, \omega^{N-1}\}$. If only linear combinations with coefficients from the ring of integers are allowed, we obtain a subring $Z[\omega] \subset Q(\omega)$ called the *ring of algebraic integers.*

What is more, if M is chosen such that $M = 2^m$, $m \geq 2$, the ring $Z[\omega]$ is *dense* in the complex plane [23]. This suggests that $Z[\omega]$ may be useful for approximating the complex numbers.

As an example, if base field F is the rationals, Q, and letting the irreducible polynomial (over the rationals) be $p(x) = x^4 + 1$, the cyclotomic polynomial of degree 4 with root $\omega = e^{2\pi j/8}$, we obtain an extension $Q(\omega)$ of the rationals of degree 4 with

basis $\{1, \omega, \omega^2, \omega^3\}$. If we restrict the linear combinations to have coefficients from the ring of integers, we obtain the ring of algebraic integers $Z(\omega) \subset Q(\omega)$ which is dense in the complex plane.

Definition 8.2. For an even integer L, the finite subset $Z[\alpha]_L$ is the finite subset

$$Z[\alpha]_L := \{a_0 + a_1\alpha + \cdots + a_{n-1}\alpha^{n-1} \mid -L/2 \leq a_i \leq L/2\}.$$

Furthermore, $Z[\alpha]_L$ is a set with $(L+1)^{n-1}$ elements.

For example, let $\alpha = e^{2\pi j/8}$ and let $L = 6$. The irreducible polynomial for α is $p(\alpha) = \alpha^4 + 1$. It is easy to see that the set $Z[e^{2\pi j/8}]_6$ is a subset of the complex plane with $7^6 = 2401$ elements. The set is shown in Fig. 8.1. Notice that the set is denser near the origin than it is at is perimeter. This suggests that the achievable quantization is finer for small complex numbers.

Another example is provided by letting $\alpha = e^{2\pi j/16}$, an element with irreducible polynomial $p(\alpha) = \alpha^8 + 1$. The set $Z[e^{2\pi j/16}]_2$ contains $3^8 = 6561$ elements. The set is also shown in Fig. 8.1. Observe the increase in the density around the origin over the previous example. For complex numbers near the origin, a very good approximation can be made by an element of $Z[e^{2\pi j/16}]_2$, in which the eight coefficients of the approximation can be represented with only *two* bits!

Assume now that we know how to quantize complex numbers into algebraic integers so that computations will occur in the set $Z[\alpha]_M$. The elements of this set correspond in an obvious way to the elements of the ring $Z[\alpha]/(M)$.

Theorem 8.1. Let $M = p_1 \cdot p_2 \cdots p_r$ be a product of distinct primes. Then the map

$$\pi : Z[\alpha]/(M) \rightarrow Z[\alpha]/(p_1) \oplus \cdots \oplus Z[\alpha]/(p_r)$$

is an isomorphism

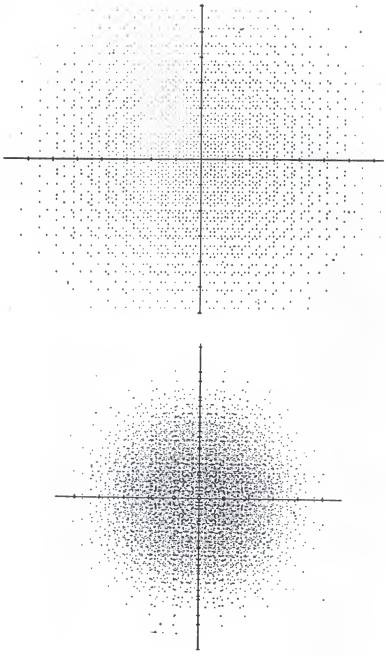


Figure 8.1. The sets $Z[e^{2\pi j/8}]_6$ (top) and $Z[e^{2\pi j/16}]_2$ (bottom)

Proof: The set of ideals $\{p_1 Z[\alpha], \ldots, p_r Z[\alpha]\}$ are clearly pairwise relatively prime. Thus, the theorem is proved by the Chinese Remainder Theorem.

Recall the isomorphism

$$Z[\alpha] \cong Z[x]/(m(x)),$$

where $m(x)$ is the minimal polynomial of degree n for α. The isomorphism is given by

$$\sum_{i=0}^{n-1} a_i \alpha^i \rightarrow f(x) := \sum_{i=0}^{n-1} a_i x^i.$$

If $f(x)$ is any element of $Z[x]/(m(x))$, we can define the polynomial

$$\bar{f}(x) := \sum_{i=0}^{n-1} (a_i \bmod p) x^i \in Z_p[x]/(m(x))$$

by reducing all of the coefficients of $f \bmod p$. Thus, we can define an isomorphism between $Z[\alpha]/(p)$ and $Z_p[x]/(m(x))$ by [20]

$$\bar{f}(\alpha) = f(\alpha) \bmod p \rightarrow \bar{f}(x).$$

This isomorphism allows us to decompose $Z[\alpha]/(p)$ by factoring $m(x)$ over Z_p.

Theorem 8.2. Let p be a prime integer and let

$$m(x) = \prod_{i=1}^{q} h_i(x)$$

be the prime factorization (over Z_p) of $m(x)$. Then

$$Z[\alpha]/(p) \cong Z_p[x]/(h_1(x)) \oplus \cdots \oplus Z_p[x]/(h_q(x)).$$

The proof follows immediately from the CRT.

The ring $Z[\alpha]/(M)$ can now be decomposed completely if we know the factorization of $m(x)$ over Z_p This is provided by the following.

Theorem 8.3. Let p be a prime integer and let $m(x) = C_R(x)$, the $R-th$ cyclotomic polynomial. If k is the smallest integer such that $p^k \equiv 1 \bmod R$, then $m(x)$ factors over Z_p into $R/2k$ distinct, irreducible polynomials $h_i(x)$ of degree k. Furthermore, if $k = 1$ and ω is a primitive $R-th$ root of unity in Z_p, then

$$C_R(x) = (x - \omega)(x + \omega)(x - \omega^3) \cdots (x + \omega^{R/2-1}).$$

In particular, we are interested in the case where $R = 2^n$ for some n. In this case, the cyclotomic polynomial $C_R(x) = x^{R/2} + 1$ [15]. To assist with the factorization, we need the following result from number theory [31].

Corollary 8.1. Consider the congruence $x^N + 1 = 0 \bmod m$. If the prime factorization of m is given by $m = p_1^{e_1} \cdots p_n^{e_n}$, then the congruence has N distinct roots if and only if N divides $(p_i - 1)/2$, $i = 1, \ldots, n$.

To relate the corollary to the factorization problem, assume that the modulus m is simply a prime p. In addition, the degree N is the integer $R/2 = 2^{n-1}$. Hence, assume that p is chosen so that $N = 2^{n-1}$ divides $(p-1)/2$, *i.e.*, $p = s \cdot 2^n + 1$ for some integer s. Then $p^1 \equiv 1 \bmod R$ and the theorem asserts that the congruence $x^{R/2} + 1 = 0 \bmod p$ will have $N = R/2$ distinct factors of degree $k = 1$.

The above is summarized by the following theorem [20].

Theorem 8.4. Let $M = p_1 \cdot p_2 \cdot p_r$ be a product of distinct primes chosen so that N divides $(p_i - 1)/2$, $i = 1, \ldots, r$, where N is the degree of the minimal polynomial for α. Then

$$Z[\alpha]/(M) \cong \bigoplus_{i=1}^{r} \bigoplus_{j=1}^{N} GF(p_i).$$

The theorem tells us that the algebraic integers can be manipulated in a manner analogous to the RNS, only with *two* levels of parallelism. At the first level, regard the algebraic integer as a polynomial. Given a set of relatively prime moduli $p_1, \ldots, p_r$, we reduce each coefficient of the algebraic integer (polynomial) modulo p_i. What is more, α satisfies the relation $m(\alpha) = 0$, so the multiplication of two approximations in the above scheme can be viewed as occurring over the ring $Z_{p_i}[x]/(m(x))$. Furthermore, if the p_i are chosen so that $m(x)$ splits into distinct linear factors over Z_{p_i}, the results from the PRNS can be used to simplify the multiplication. This is the second level of parallelism. The combination of the extension field and PRNS is called the Algebraic Integer RNS (AIRNS) [23]. See figures 2 and 3 for diagrams of the derived architectures.

Let us consider the case where $m(x) = x^4 + 1$ in depth. Assume that p_i is chosen so that the congruence $x^4 + 1 = 0 \bmod p_i$ has 4 distinct roots r_0, r_1, r_2, r_3. It can be shown [31] that these roots are additive and multiplicative inverses of one another. Thus, if we let $r := r_0$, then $r_1 = -r$, $r_2 = r^{-1}$, *and* $r_3 = -r^{-1}$. The polynomial RNS provides the isomorphism ϕ between $Z_{p_i}[x]/m(x)$ and $Z_{p_i}^4$ which can be represented as a matrix multiplication. Formally, associate with an algebraic integer the 4-tuple of coefficients

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3 \sim (a_0, a_1, a_2, a_3)^T := A(x)$$

and map the algebraic integer to a 4-tuple in $Z_{p_i}^4$

$$A'(x) := (a'_0, a'_1, a'_2, a'_3) = QA(x)$$

where

$$Q = \begin{pmatrix} 1 & r & r^2 & r^3 \\ 1 & -r & r^2 & -r^3 \\ 1 & -r^3 & -r^2 & -r \\ 1 & r^3 & -r^2 & r \end{pmatrix}.$$

The mapping from the AIRNS to the complex numbers is straightforward; it is just a linear combination of the powers of the primitive root of unity. The problem of approximating complex numbers by algebraic integers, which is required as the first step in AIRNS processing, is more difficult. The problem has been studied by Games [22, 21], and his solution is based on continued fraction approximations. This solution, while theoretically elegant, does not appear to have a real-time implementation with a bandwidth that can meet the needs of complex signal processing applications.

A complete AIRNS system requires a unit to perform the mapping (or approximation) from the complex field to the ring of algebraic integers, a forward PRNS mapping unit, finite field multipliers to perform the AIRNS computations, an inverse PRNS mapping unit, and an algebraic integer-to-complex mapping unit. All of this is required in order to perform complex arithmetic where the only gained benefit is the avoidance of scaling. The difficulty of complex approximation by algebraic integers, coupled with the high hardware overhead, do not seem to justify the savings in eliminating scaling, which we have shown is no longer a difficult operation. Hence, we will not consider the AIRNS further.

CHAPTER 9
DISTRIBUTED ARITHMETIC IMPLEMENTATION OF FFTs

9.1 The Good-Thomas FFT

The Good-Thomas FFT algorithm is similar to the Cooley-Tukey FFT but relies on a special permutation of the data to achieve a structure which is twiddle factor free. To begin, assume that the sequence $\{x(i)\}$ is of length n where $N = p_1 p_2$, the product of two relatively prime integers. Recall that the DFT of $\{x(i)\}$ is given by

$$X(k) = \sum_{i=0}^{N-1} W_N^{ik} x(i)$$

where

$$W_n = e^{-2\pi j/N}.$$

Define the parameters i_1 and i_2 by

$$i_1 = i \bmod p_1 \tag{9.1}$$

and

$$i_2 = i \bmod p_2. \tag{9.2}$$

From the section on the Chinese remainder theorem, it follows that

$$i = (m_1 n_1 i_1 + m_2 n_2 i_2) \bmod N$$

where $m_1 = p_2$, $m_2 = p_1$, $n_1 = m_1^{-1} \bmod p_1$, and $n_2 = m_2^{-1} \bmod p_2$. Also, define the parameters k_1 and k_2 by

$$k_1 = n_1 k \bmod p_1 \tag{9.3}$$

and

$$k_2 = n_2 k \bmod p_2. \tag{9.4}$$

It then follows that

$$k = (m_1 k_1 + m_2 k_2) \bmod N. \tag{9.5}$$

Rewrite the expression for the DFT using the indices i_1, i_2, k_1, and k_2:

$$X(m_1 k_1 + m_2 k_2) = \sum_{i_1=0}^{p_1-1} \sum_{i_2=0}^{p_2-1} W_N^{(m_1 n_1 i_1 + m_2 n_2 i_2)(m_1 k_1 + m_2 k_2)} x(m_1 n_1 i_1 + m_2 n_2 i_2).$$

But $W_N^{m_1 m_2 n_2 i_2 k_1} = W_N^{m_1 m_2 n_1 i_1 k_2} = 1$ so the cross- terms disappear. For brevity, refer to the sequences X and x by their two dimensional indices as x_{i_1,i_2} and X_{k_1,k_2}. Then

$$X_{k_1,k_2} = \sum_{i_1=0}^{p_1-1} \sum_{i_2=0}^{p_2-1} W_N^{m_1^2 n_1 i_1} W_N^{m_2^2 n_2 i_2} x_{i_1,i_2}.$$

Let

$$\beta := W_N^{m_1^2 n_1}$$

and

$$\gamma := W_N^{m_2^2 n_2}.$$

Since $\beta = (W_N^{m_1})^{m_1 n_1}$, $W_N^{m_1} = e^{-2\pi j/p_1}$, and $m_1 n_1 = 1 \bmod p_1$, it follows that

$$\beta = W_{p_1}.$$

Similarly,

$$\gamma = W_{p_2}$$

and hence

$$X_{k_1,k_2} = \sum_{i_1=0}^{p_1-1} \sum_{i_2=0}^{p_2-1} W_{p_1}^{i_1 k_1} W_{p_2}^{i_2 k_2} x_{i_1,i_2}. \tag{9.6}$$

What is significant about Eq. 9.6 is that the DFT of the sequence, when permuted, can be computed as a true two- dimensional DFT with no twiddle factors. In practice,

the input sequence is sorted into a two-dimensional array with the array indices, i_1 and i_2 given by the index's residues modulo the relatively prime factors of N. The FFT points are recovered from the array according to the rules in Eqs. 9.3 and 9.4.

For example, suppose $n = 15 = 3 \cdot 5$. The CRT parameters are $m_1 = 5, n_1 = 2$ and $m_2 = 3, n_2 = 2$. According to Eqs. 9.1 and 9.2, the input sequence $\{x(k)\}$ is placed into a two-dimensional array as follows:

$$\begin{pmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & x_{0,4} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \end{pmatrix} = \begin{pmatrix} x(0) & x(6) & x(12) & x(3) & x(9) \\ x(10) & x(1) & x(7) & x(13) & x(4) \\ x(5) & x(11) & x(2) & x(8) & x(14) \end{pmatrix}$$

A five-point DFT is performed on each of the rows of the matrix resulting in the intermediate matrix

$$\begin{pmatrix} \tilde{x}_{0,0} & \tilde{x}_{0,1} & \tilde{x}_{0,2} & \tilde{x}_{0,3} & \tilde{x}_{0,4} \\ \tilde{x}_{1,0} & \tilde{x}_{1,1} & \tilde{x}_{1,2} & \tilde{x}_{1,3} & \tilde{x}_{1,4} \\ \tilde{x}_{2,0} & \tilde{x}_{2,1} & \tilde{x}_{2,2} & \tilde{x}_{2,3} & \tilde{x}_{2,4} \end{pmatrix}.$$

A three-point DFT is then performed on each of the columns which leads to the matrix

$$\begin{pmatrix} X_{0,0} & X_{0,1} & X_{0,2} & X_{0,3} & X_{0,4} \\ X_{1,0} & X_{1,1} & X_{1,2} & X_{1,3} & X_{1,4} \\ X_{2,0} & X_{2,1} & X_{2,2} & X_{2,3} & X_{2,4} \end{pmatrix}.$$

Using Eq. 9.5 to unscramble the harmonic indices from the array indices k_1 and k_2, the DFT points are sitting in the matrix in the order

$$\begin{pmatrix} X(0) & X(3) & X(6) & X(9) & X(12) \\ X(5) & X(8) & X(11) & X(14) & X(2) \\ X(10) & X(13) & X(1) & X(4) & X(7) \end{pmatrix}.$$

9.2 The Rader Prime Algorithm

Recall that the Rader prime algorithm converts a prime-blocklength DFT into a computation of the form

$$X(\pi^l) = x(0) + \sum_{j=0}^{p-2} W_p^{\pi^{l-j}} x(\pi^{p-1-j}). \tag{9.7}$$

The sum is recognized to be a cyclic convolution of the sequences $\{W_p^{\pi^j}\}$ and $\{x(j')\}$.

For example, to compute a five-point DFT using the Rader algorithm, $W_p = e^{-j2\pi/5}$ and

$$X(0) = \sum_{i=0}^{4} x(i).$$

The indices are over Z_5 and using 2 as a primitive element, use the permutation rule

$$\{2^0, 2^1, 2^2, 2^3\} = \{1, 2, 4, 3\}$$

so that the Rader prime algorithm is defined as the 4-point cyclic convolution of the sequences

$$\{W_p^{\pi^j}\} = \{W_p, W_p^2, W_p^4, W_p^3\}$$

and

$$\{x(\pi^{p-1-j})\} = \{x(1), x(3), x(4), x(2)\}.$$

to produce

$$\{(X(\pi^l) - x(0))\} = \{(X(1) - x(0)), (X(2) - x(0)), (X(4) - x(0)), (X(3) - x(0))\}.$$

In practice, the d.c. term $X(0)$ is computed as the sum

$$X(0) = \sum_{i=0}^{4} x(i)$$

and the nonzero harmonics are computed by first computing the sum in Eq. 9.7 and then adding the term $x(0)$.

9.3 Distributed Arithmetic

Consider the calculation of the inner product

$$y = \sum_{i=0}^{n-1} w_i x_i$$

where the w_i are known *a priori* and the wordlength of the data x_i is limited to b bits. Then x_i can be represented as

$$x_i = \sum_{j=0}^{b-1} 2^j x_i^{(j)}$$

where $x_i^{(j)}$ is the j-th bit in the binary expression for x_i. Then

$$y = \sum_{i=0}^{n-1} \sum_{j=0}^{b-1} w_i 2^j x_i^{(j)}.$$

Reversing the order of summation,

$$y = \sum_{j=0}^{b-1} 2^j \sum_{i=0}^{n-1} w_i x_i^{(j)}.$$

The inner summation can be interpreted as a function of the j-th common bits of the words x_i which will be denoted by

$$\sum_{i=0}^{n-1} w_i x_i^{(j)} =: \Phi(\vec{x}^{(j)})$$

so that

$$y = \sum_{j=0}^{b-1} 2^j \Phi(\vec{x}^{(j)}).$$

This is simply a shift-accumulate operation of the functions Φ computed at each bit location of the x_i.

In practice, the function $\Phi(\vec{x}^{(j)})$ can be computed by using the j-th common bits of the x_i as an address to a table which stores the precomputed values of Φ. This requires a table with an address space of 2^n and wordwidth equal to the number of bits in the binary representation of the w_i. For a given j, the value of Φ is looked up, shifted j bits to the left, and added to the partial result. This takes a total of b clock cycles.

9.4 The Distributed Arithmetic Small FFT

Suppose, as in the earlier section, that we wish to perform a $3 \cdot 5 = 15$ point Good-Thomas FFT. Arrange the data in the two-dimensional array indexed with the indices i_1 and i_2:

$$\begin{pmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & x_{0,4} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \end{pmatrix}.$$

A distributed arithmetic version of the Rader algorithm will be used to perform the row and column transforms by cyclic convolution.

Notice first that the cyclic convolution can be implemented by storing the sequence in a cyclic shift register and keeping the transform coefficients inplace. Also, since the zeroth sequence term is given special significance in the Rader algorithm, the zeroth terms of each row (which correspond to the first column of the permuted data matrix) are stored in their own cyclic shift register. The rest of the matrix is stored in a second cyclic shift register in column order. Every fourth word of the shift register is connected to the table. Refer to Fig. 9.1 for a schematic of the distributed arithmetic architecture.

As a matter of notation, the symbol x_{0,i_2} means the zeroth row of the matrix x. To begin, the row $x^{(0)}_{0,i_2}$ is sent to the first table which computes the first partial product for $\tilde{x}_{0,0}$. This partial product is stored on a stack and its lsb (which happens to equal $\tilde{x}^{(0)}_{0,0}$) is sent to a three-bit buffer. Next, the registers perform a cyclic shift and row $x^{(0)}_{1,i_2}$ is sent to the first table which computes the first partial product for $\tilde{x}_{1,0}$. This product is stored on a stack and $\tilde{x}^{(0)}_{1,0}$ is sent to the three-bit buffer. Another cyclic shift is performed, and row $x^{(0)}_{2,i_2}$ is sent to the table, which computes the first partial product for $\tilde{x}_{2,0}$. This product is stored on a stack and $\tilde{x}^{(0)}_{2,0}$ is sent to the three-bit buffer. At this point, $\tilde{x}^{(0)}_{i_1,0}$ is sitting in the three-point buffer.

At this time, the next cyclic shift delivers row $x_{0,i_2}^{(0)}$ in permuted order to the first table. This has the effect of computing the partial product for one of the other harmonics in the row, say $\tilde{x}_{0,1}$. The process of cyclic shifting, writing to the stack, and sending $\tilde{x}_{1,1}^{(0)}$, $\tilde{x}_{2,1}^{(0)}$ to the three-bit buffer is repeated, which results in the computation of $\tilde{x}_{i_1,1}^{(0)}$.

While $\tilde{x}_{i_1,1}^{(0)}$ was being computed, the previous contents of the three-bit buffer, $\tilde{x}_{i_1,0}^{(0)}$, were sent to three smaller distributed arithmetic units which performed all of the permutations necessary to calculate the first partial products of $X_{k_1,0}$, which were written to their own stacks.

The process continues until the cyclic shift results in the data being in its original state. At this point, all of the first partial products for $\tilde{x}_{i_1,i_2}$ and X_{k_1,k_2} have been computed and loaded onto the stacks. Next, the row $x_{0,i_2}^{(1)}$ is sent to the first table, the second partial product for $\tilde{x}_{0,0}$ is computed and added to the first partial product from the stack, and the sum is written to the stack. Now, $\tilde{x}_{0,0}^{(1)}$ is sent to the three-bit buffer. The process is continued as before until all of the second partial products for $\tilde{x}_{i_1,i_2}$ and X_{k_1,k_2} are computed. The data will continue to be circularly shifted and moved down by one bit each time the original ordering occurs until the final common bits of the rows are met. The process is summarized as follows:

for $r = 0$ **to** r **=wordlength:**

$$\left.\begin{array}{l} x^{(r)}_{0,i_1} \longrightarrow \tilde{x}^{(r)}_{0,0} \\ x^{(r)}_{1,i_1} \longrightarrow \tilde{x}^{(r)}_{1,0} \\ x^{(r)}_{2,i_1} \longrightarrow \tilde{x}^{(r)}_{2,0} \end{array}\right\} = \tilde{x}^{(r)}_{i_1,0} \longrightarrow X^{(r)}_{k_1,0}$$

$$\left.\begin{array}{l} x^{(r)}_{0,i_1} \longrightarrow \tilde{x}^{(r)}_{0,1} \\ x^{(r)}_{1,i_1} \longrightarrow \tilde{x}^{(r)}_{1,1} \\ x^{(r)}_{2,i_1} \longrightarrow \tilde{x}^{(r)}_{2,1} \end{array}\right\} = \tilde{x}^{(r)}_{i_1,1} \longrightarrow X^{(r)}_{k_1,1}$$

$$\left.\begin{array}{l} x^{(r)}_{0,i_1} \longrightarrow \tilde{x}^{(r)}_{0,2} \\ x^{(r)}_{1,i_1} \longrightarrow \tilde{x}^{(r)}_{1,2} \\ x^{(r)}_{2,i_1} \longrightarrow \tilde{x}^{(r)}_{2,2} \end{array}\right\} = \tilde{x}^{(r)}_{i_1,2} \longrightarrow X^{(r)}_{k_1,2}$$

$$\left.\begin{array}{l} x^{(r)}_{0,i_1} \longrightarrow \tilde{x}^{(r)}_{0,3} \\ x^{(r)}_{1,i_1} \longrightarrow \tilde{x}^{(r)}_{1,3} \\ x^{(r)}_{2,i_1} \longrightarrow \tilde{x}^{(r)}_{2,3} \end{array}\right\} = \tilde{x}^{(r)}_{i_1,3} \longrightarrow X^{(r)}_{k_1,3}$$

$$\left.\begin{array}{l} x^{(r)}_{0,i_1} \longrightarrow \tilde{x}^{(r)}_{0,4} \\ x^{(r)}_{1,i_1} \longrightarrow \tilde{x}^{(r)}_{1,4} \\ x^{(r)}_{2,i_1} \longrightarrow \tilde{x}^{(r)}_{2,4} \end{array}\right\} = \tilde{x}^{(r)}_{i_1,4} \longrightarrow X^{(r)}_{k_1,4}$$

return with $r = r + 1$

If the length of the sequence is L and the data is N-bits in length, the latency form initial presentation of the sequence to the appearance of the last bit of the transform is $L \cdot N$ clock cycles plus the three clock cycles necessary to fill the three-bit buffer the first time. For a 15-point transform with 16-bit data, a 10ns clock translates into approximately 2.4 μs per transform.

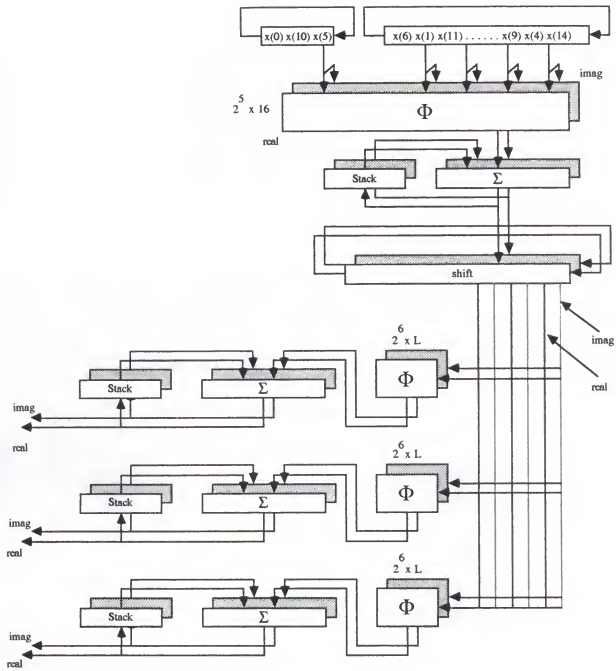


Figure 9.1. Distributed arithmetic FFT engine

CHAPTER 10
THE FFT ARRAY PROCESSOR

10.1 Introduction

This chapter details the design of an array processor for the ultra high-speed computation of fast Fourier transforms. The design of this processor draws upon most of the theory we have developed up to this point. We call our machine the FFT array processor (FFTAP).

The FFTAP is based on the QRNS. As we already have shown, the QRNS is table look-up intensive and leads to a realization for complex arithmetic primitives which is multiplier-free. Essentially, a multiplier is replaced by a binary adder. This is important since it allows us to place many multipliers in a single VLSI package, providing a significant advantage in arithmetic functionality over conventional arithmetic systems.

As a result of using the QRNS, we can create complex multiply/accumulate engines which operate as much as ten times faster than the conventional state-of-the-art while using as little as one-tenth of the area. These ALUs are highly pipelineable and have the additional property of graceful degradation. As such, the QRNS will be shown to be the medium of choice for the VLSI implementation of a high-speed FFT array processor capable of producing 512-point complex FFTs at the rate of roughly 780K transforms per second.

Since the QRNS is a fixed-point system, there have been significant problems with loss of precision. Because the QRNS is an unweighted number system, there is no acceptable manner in which to handle dynamic range overflow. For each nested multiply operation, there is a geometric growth in the dynamic range. Countering this problem requires conversion from the QRNS to the integer system, scaling, and returning to the QRNS for further processing.

Historically, scaling has been the principal limitation of QRNS system design. The required QRNS to integer conversion is slow and hardware-intensive and relies on the design of large custom modulo-M adders. The efficacy of the QRNS is limited by the number of scaling calls to manage the dynamic range and hence the QRNS's utility was typically limited to long sums-of-products. Because of this, most QRNS DFT processors depended on the use of convolution form DFTs such as the Goertzel algorithm or the Rader prime algorithm. While this did result in a realizable design, it did not take advantage of the computational efficiency provided by decimation-in-time algorithms.

Decimation-in-time FFT algorithms, however, are based on nested multiplies and hence suffer from dynamic range problems. To take advantage of the drastic reduction in multiplicative complexity afforded by such algorithms, we have extended the RNS scaling algorithm to develop a QRNS scaling algorithm which also uses only standard binary hardware and can be pipelined at the QRNS's computational speed. This moves the QRNS into a middle ground where it is capable of extremely high throughput while executing a broader class of algorithms than was previously possible.

The machine is based on a radix-8 decimation-in-time algorithm for in-place computation of a 512-point complex FFT. What makes our machine an "array" processor is that eight operands are simultaneously operated upon by a radix-8 computational engine. This allows a radix-8 butterfly computation (actually an eight-point discrete Fourier transform) to be performed completely in parallel for high-speed execution. In order to utilize the full power of this computational engine, QRNS input and output converters needed to be designed which were capable of supporting the extremely high data-conversion bandwidth required by the processor. The design of these converters and of the FFTAP memory subsystem is also quite innovative and are discussed in detail in the report.

10.2 The Radix-8 Fast Fourier Transform

10.2.1 Introduction

The discrete Fourier transform (DFT) of a (possibly complex) sequence

$$\{x(k)\},\ k = 0, 1, \ldots, N-1$$

is the complex sequence

$$X(n) = \sum_{k=0}^{N-1} x(k) W_N^{kn},\ n = 0, 1, \ldots, N-1 \tag{10.1}$$

where W_N is the primitive N-th root of unity given by

$$W_N = e^{-2\pi j/N},\ j = \sqrt{-1}.$$

Eq. 10.1, as written, requires on the order of N^2 complex multiplications and additions, which can become prohibitively large for even modest blocklengths N. There

are many approaches to the evaluation of the DFT which drastically reduce the multiplicative complexity of the computation. The algorithm we will use is called the *radix-8 fast Fourier transform* (FFT).

To derive the radix-8 FFT, assume that the blocklength N is a multiple of 8 and separate the data points into 8 disjoint sets whose indices are all congruent modulo 8. That is, rewrite Eq. 10.1 as

$$X(n) = \sum_{i=0}^{7} \sum_{r=0}^{N/8-1} x(8r+i) W_N^{(8r+i)n}$$

which can be seen to equal

$$X(n) = \sum_{i=0}^{7} W_N^{in} \sum_{r=0}^{N/8-1} x(8r+i) W_N^{8rn}.$$

It is simple to verify that

$$W_N^8 = W_{N/8}$$

so that

$$X(n) = \sum_{i=0}^{7} W_N^{in} \sum_{r=0}^{N/8-1} x(8r+i) W_{N/8}^{rn}.$$

The inner summation in the above equation is recognized as the $N/8$-point DFT of the data points whose indices are congruent modulo 8. Denoting these smaller DFTs by $S^{(i)}(n)$, $i = 0, 1, \ldots, 7$, we have

$$X(n) = \sum_{i=0}^{7} W_N^{in} S^{(i)}(n). \tag{10.2}$$

What Eq. 10.2 says is that to calculate the value of the n-th harmonic $X(n)$, we need only calculate the n-th harmonics of the sequences $S^{(i)}$ and combine them with the "twiddle factors" W_N^{in}. Furthermore, the $N/8$-point DFTs $S^{(i)}$ are periodic with period $N/8$. This means that for any harmonic $r = n + qN/8$, $q = 0, 1, \ldots, 7$, we have $S^{(i)}(n) = S^{(i)}(r)$ and only the twiddle factors change.

To assess the reduction in multiplicative complexity, there are eight $N/8$-point DFTs, each requiring $(N/8)^2$ multiplications for a total of $N^2/8$ multiplications. Additionally, for each output harmonic, there are eight twiddle factor multiplications, which brings the total multiply count to $N^2/8 + 8N$. Although this might not seem like a substantial savings, consider the case where $N/8$ is itself a multiple of eight. The same "decimation" procedure can be performed on the $N/8$-point DFTs $S^{(i)}$. If N is a power of eight, the DFT can be decimated a total of $\log_8 N$ times until arriving at a level of eight-point transforms.

To obtain the values of the harmonics with index greater than $N/8$, break the harmonic indices into two subindices: one "coarse" and one "vernier":

$$n + lN/8, \quad q = 0, 1, \ldots, 7 \quad \text{and} \quad n = 0, 1, \ldots, N/8 - 1.$$

Then Eq. 10.2 can be rewritten as

$$X(n + lN/8) = \sum_{i=0}^{7} W_N^{i(n+lN/8)} S^{(i)}(n + lN/8).$$

But $S^{(i)}$ is $N/8$-periodic so that

$$S^{(i)}(n + N/8) = S^{(i)}(n).$$

Also,

$$W_N^{N/8} = W_8$$

so that

$$X(n + lN/8) = \sum_{i=0}^{8} W_8^{li} (W_N^{in} S^{(i)}(n)). \tag{10.3}$$

What Eq. 10.3 says is that to compute all of the harmonics corresponding to the p-th index congruence class modulo $N/8$, we need to premultiply the p-th element

of each sequence $S^{(i)}$ by W_N^{ip} and perform the eight point DFT. In other words, the eight harmonics in the congruence class modulo p are the DFT points of the sequence $\{W_N^{ip} S^{(i)}(p)\}$. Recursively, the transform is broken down into $N/8$ eight-point transforms, the twiddle factors are multiplied, $N/8$ new eight-point transforms are computed, and so on until the final stage.

If $N/8$ is a multiple of 8, then the $S^{(i)}$ can be broken down into nested sequences of transfroms $T^{(iq)}$ and pieced together in a similar fashion. The general case is given by

$$S^{(i)}(n + mN/64) = \sum_{q=0}^{7} W_8^{mq} \{W_{N/64}^{nq} T^{(iq)}(n)\}$$

where

$$T^{(iq)} = \sum_{p=0}^{7} x(8(8p+q)+i) W_{N/64}^{np} = \sum_{p=0}^{7} x(64p+8q+i) W_{N/64}^{np}.$$

For the case $N = 512$, the final decomposition is

$$X(n) = \sum_{i=0}^{7} W_8^{li} \left\{ W_{512}^{ni} \left(\sum_{q=0}^{7} W_8^{mq} \left(W_{64}^{nq} \sum_{p=0}^{7} x(64p+8q+i) W_8^{np} \right) \right) \right\}. \qquad (10.4)$$

Eq. 10.4 clearly illustrates how the 512-point radix-8 FFT is computed. There are three levels of nested eight-point transforms. The inner-most eight-point transform is computed with indices i and q held fixed and p varying. The index of the sequence x is given by $64p + 8q + i$, which corresponds to "three-bit reversal". Thus, if the sequence x is first rearranged in three-bit reversed order, the data can first be accessed sequentially. This is explained in depth in the next section.

If N is a power of 8, there are a total of $\log_8$ stages of decimation. Suppose we can compute an eight-point DFT with M multiplications. At each stage, there are $N/8$ eight-point DFTs followed by N twiddle factor multiplications. Thus, for each stage there are $(N/8)M + N$ multiplications which yields a total of $((N/8)M + N) \log_8 N$

multiplications. If the eight-point DFTs are evaluated directly, requiring $M = 64$ multiplications, the complexity is $9N \log_8 N$. A further reduction can be made in the overall complexity by using a fast algorithm to compute the eight-point transforms, which is used in our design. We use a radix-2 decimation-in-time algorithm which uses $M = 16$ multiplications and gives an overall complexity of $3N \log_8 N$.

10.2.2 Efficient Memory Addressing for Parallel Computation of the Radix-8 FFT

One of the desirable attributes of the radix-2 FFT is the ability to perform *in-place computations.* That is, the sequence can be Fourier transformed by removing two pieces of data at a time, performing a radix-2 butterfly computation, and then returning the transformed data to their original storage locations. This eliminates the need for additional memory to store intermediate results. We will show that a similar in-place scheme can be developed for the radix-8 transform.

In a radix-2 FFT, efficient addressing of the data is facilitated by permuting its indices by *bit reversal.* If the binary representation of the i-th data point is

$$i = b_{n-1} \cdots b_1 b_0,$$

form the new index i' by reversing the binary digits:

$$i' = b_0 b_1 \cdots b_{n-1}.$$

The sequence $\{x(i)\}$ is stored in memory as $\{x(i')\}$ and the data are now in the proper order to be delivered to the first stage of the radix-2 FFT for in-place computation.

It would be fortunate if there were a similar method of permuting indices for the radix-8 FFT. To achieve this goal, suppose the blocklength is $8^3 = 512$ data points. The first stage of decimation breaks the data points into eight 64-point transforms. The data points in the first 64-point transform have indices which are all multiples

of eight. The data points in the second 64-point transform have indices which have remainder one when divided by eight, and so on. The second stage of decimation takes each 64-point transform and breaks it into eight 8-point transforms. The data partitioning in this second stage of decimation is analogous to the first stage. For example, the first 64-point transform is broken into eight 8-point transforms; the first consists of elements 0,8,16,24,32,40,48, and 56 of the first 64-point sequence. The second consists of elements 1,9,17,25,33,41,49, and 57 of the first 64-point sequence.

Upon closer examination, it can be seen that if the data is *three-bit* reversed, the sequence is delivered in proper order to the first stage of eight-point transforms. That is, let the binary representation of the i-th index be

$$i = c_2c_1c_0 \;\; b_2b_1b_0 \;\; a_2a_1a_0.$$

Form the new index i' according to

$$i' = a_2a_1a_0 \;\; b_2b_1b_0 \;\; c_2c_1c_0 \tag{10.5}$$

and reorder the sequence as $\{x(i')\}$. The data are now in the proper order to be delivered to the first stage of eight-point transforms. Furthermore, the eight transformed points can be returned to the same set of storage locations used to store the untransformed data. This is because the data will not be used again until the entire first stage of eight-point transforms has been computed.

Suppose now that we are transforming the sequence $\{x(i')\}$. It is readily apparent that the transforms in the first stage of the FFT are performed on data points with indices separated by one. The transforms in the second stage use data points whose indices are separated by 8, and the final stage uses data points whose indices are separated by 64. It will be shown that a particularly elegant way to store the data

is in a "cubic" memory in which eight operands can be fetched along one of three coordinate directions.

10.2.3 Doubling FFT

If we are working with real input sequences, the DFT of two sequences of length N can be computed simultaneously. This also allows transforms of sequences of length $2N$ to be computed.

Suppose f and g are two real N-point sequences. Form the complex sequence h as

$$h(k) = f(k) + jg(k).$$

By the linearity of the DFT,

$$H(n) = F(n) + jG(n).$$

Because of the periodicity and symmetry properties of the DFT, we have

$$h^*(k) \longrightarrow H^*(N-n)$$

where

$$H^*(N-n) = F^*(N-n) - jG^*(N-n) = F(n) - jG(n).$$

It then follows that

$$F(n) = \frac{H(n) + H^*(N-n)}{2}$$

and

$$G(n) = \frac{H(n) - H^*(N-n)}{j2}.$$

The above can be used to compute the DFT of a real sequence of length $2N$. Suppose $\{x(k)\}$ is real of length $2N$. Define the length N sequences f and g by

$$f(k) = x(2k)$$

and

$$g(k) = x(2k+1).$$

With $h(k)$ defined by $h(k) = f(k) + jg(k)$, compute $F(n)$ and $G(n)$ as above. Then the DFT of x is computed as

$$\begin{aligned} X(n) &= F(n) + W_{2N}^{n} G(n), \qquad n = 0, 1, \ldots, N-1 \\ X(n+N) &= F(n) - W_{2N}^{n} G(n), \quad n = 0, 1, \ldots, N-1. \end{aligned}$$

10.3 Efficient CRT Implementation

In practice, we can apply our scaling CRT and incorporate the QRNS to CRNS conversions directly into the tables. There will be two scaling CRTs performed simultaneously: one for the real and one for the imaginary part. The tables in the real CRT can be preceded by an adder to produce the sum $(z_i + z_i^*)$. The real tables now perform the function

$$\alpha_i = [[m_i \left(n_i 2_i^{-1} (z_i + z_i^*) \bmod p_i \right) / d]].$$

The tables in the imaginary CRT can be preceded by a subtractor to produce $(z_i - z_i^*)$ and the imaginary tables perform the function

$$\gamma_i = [[m_i \left(n_i (2_i \hat{j}_i)^{-1} (z_i + z_i^*) \bmod p_i \right) / d]].$$

In this case, the output of the complex scaled CRT is $X_s + jY_s$, where

$$X_s = \left(\sum_{i=1}^{L} \alpha_i \bmod 2^k \right) \bmod 2^k$$

and

$$Y_s = \left(\sum_{i=1}^{L} \gamma_i \bmod 2^k \right) \bmod 2^k.$$

The parameters d and k are as described in the section on the scaled CRT.

Recall that the signed RNS maps positive to the lower half of the dynamic range and negative numbers to the upper half. Typically, this represents a difficulty in interfacing with conventional systems since the magnitude of a negative number is given by $M - |x|$. The scaled CRT, however, uses a reduced dynamic range which is a power of 2. Thus, the negative numbers are given by $2^n - |x|$, which is the standard two's-complement form for a negative number x.

Symbolically, we can represent the conversion from QRNS pairs to scaled complex integers by the mapping Π_M where

$$\Pi_M((z_1, z_1^*), (z_2, z_2^*), \ldots, (z_8, z_8^*)) = X_s + jY_s.$$

A diagram of the scaled CRT for QRNS output conversion is shown in Fig. 10.1

10.4 The FFT Array Processor

10.4.1 Introduction

This chapter explains the operation of the FFT array processor (FFTAP) and details all of the subsystems which it comprises. The FFTAP is a QRNS-based machine which is capable of computing FFTs at high bandwidths in a limited physical volume with very little power dissipation. The FFTAP owes these attributes to the QRNS which also endows it with a certain amount of fault tolerance and graceful degradation.

The FFTAP is predicated upon the use of full-custom VLSI for the design of all of the component subsystems. At first glance, this may seem like a substantial undertaking. Upon further examination, however, it will be seen that there are only *three* cells from which *all* of the required hardware can be built: small tables, eight-bit

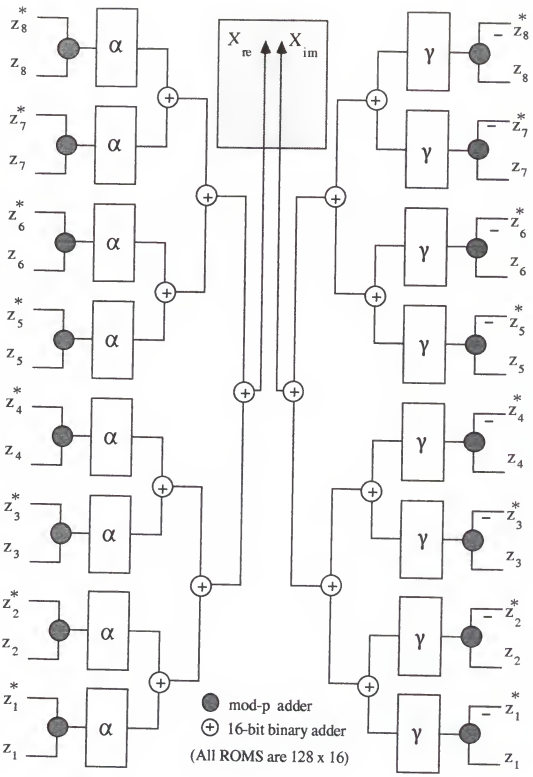


Figure 10.1. Scaled CRT engine for QRNS output conversion

binary adders, and eight-bit latches. There exits an enormous degree of repetitivity at both the microscopic (chip internal) and macroscopic (board-level).

The internal repetitivity within the VLSI chips allows for extremely simple layout with a minimum of interconnection. This is important from a VLSI standpoint since it enables many elements to be packed densely onto a single chip, which greatly enhances functionality and reduces the need to go off-chip. As such, the throughput necessary to realize a megahertz-class FFT machine can be achieved.

The same type of repetitivity and modularity also exists at the board-level which allows for ease of board layout with a minimum of communication between chips. As will be seen, this enables the FFTAP to be cascadable, which can actually *triple* the throughput. Ease of interconnection between chips is significant since it helps minimize the transmission-line effects that tend to occur when running pc boards at high clock rates.

Finally, an FFTAP design is ultimately envisioned which is RAM based. As such, there are not only three distinct cell types; there are only three distinct *chip* types. Fabrication costs will then be reduced significantly, making the construction of an FFTAP economically feasible.

10.4.2 Overview of the FFTAP

The FFTAP is capable of computing 512-point complex FFTs or inverse FFTs at extremely high bandwidths. The internal numeric representation is the QRNS (detailed in Chapter 3) and the FFT is implemented as a radix-8 transform (detailed in Chapter 2).

It was shown that a 512-point radix-8 FFT is computed as three stages of 64 eight-point DFTs, with each stage preceded by a sequence of twiddle factor multiplications.

The FFTAP computes the eight-point DFTs by using a radix-2 decimation-in-time FFT to accelerate the computation time. Furthermore, the eight-point FFT and associated twiddle factor multiplications are computed in parallel. That is, eight operands are delivered simultaneously to the QRNS radix-8 engines by the QRNS input conversion subsystem.

The radix-8 engines and input conversion elements are highly pipelined so that upon delivery of the first eight operands, the next eight operands can be delivered in the immediately following clock cycle. As the operands are passed out of the radix-8 engines, they are returned to complex integer format by the scaled CRT subsystem and returned to memory for processing at the next stage of eight-point FFTs.

The scaled CRT subsystem is also pipelined so that it can process QRNS to complex conversions at the throughput rate of eight operands per clock cycle. More significantly, there is natural pipelining between all three of the component subsystems so that a sustained throughput of eight operands per clock cycle can be maintained. Hence, not including initial latency, each stage of the radix-8 FFT requires 64 clock cycles, and there are three such stages which translates into 192 clock cycles per 512-point FFT (if the system is run in standalone mode). Assuming a 20 ns. clock, in this mode the FFTAP is capable of a sustained throughput of 260K transforms per second. This is tripled in cascade mode for a total of 780K transforms per second. A block diagram of the FFTAP is shown in Fig. 10.2.

10.4.3 Input Conversion Subsystem

The input conversion subsystem must deliver eight operands to each of the radix-8 engines operating in each QRNS channel. There are a total of eight QRNS channels,

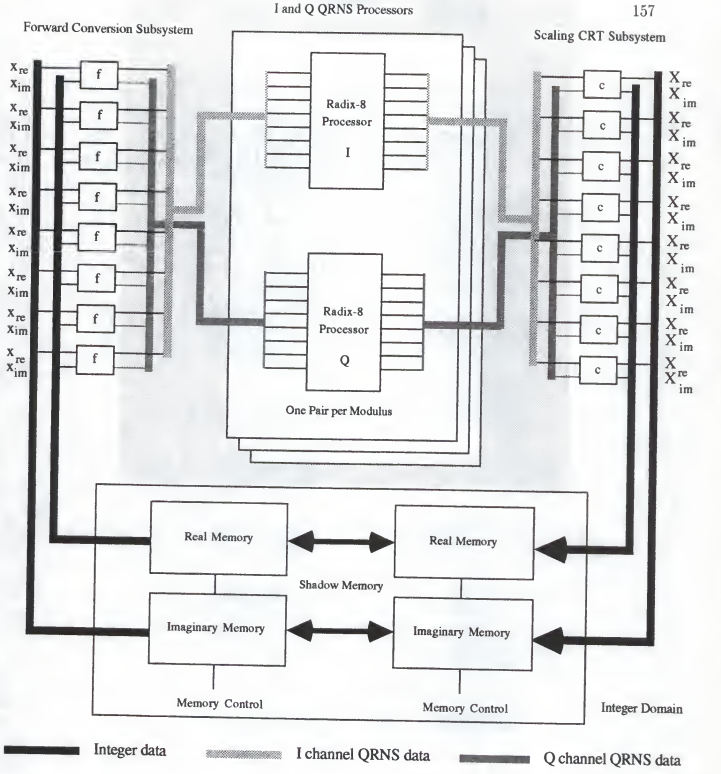


Figure 10.2. Block diagram of FFTAP

each comprised of two subchannels which we will call I and Q subchannels (actually, an abuse of nomenclature).

The input conversion problem is partitioned as follows. For each complex operand, a QRNS input conversion element performs the mapping ϕ_M, which is a composition of the mappings ϕ_{p_i}, each of which deliver I and Q information to one of the eight modulus channels. Thus, ϕ_M maps a complex operand to the complete set of QRNS pairs corresponding to that operand. There are eight such QRNS conversion elements performing mappings we will denote by

$$\phi(l)_M = (\phi(l)_{p_1}, \phi(l)_{p_2}, \ldots, \phi(l)_{p_8}), \quad l = 1, 2, \ldots, 8.$$

A schematic of one QRNS conversion element was shown in Fig. 4.1. The forward conversion is pipelined so that a new operand can be delivered to the element each clock cycle. A chip in the input conversion subsystem consists of eight such elements and is shown in Fig. 10.3. After the initial latency, all eight of the QRNS pairs corresponding to an input operand are delivered each clock cycle. There are eight chips in the QRNS conversion subsystem, one each corresponding to the mapping $\phi(l)_M$, $l = 1, 2, \ldots, 8$. The eight chips run in parallel and can accept a group of eight complex operands per clock cycle. After the initial latency, all of the QRNS pairs for all eight input operands are delivered each clock cycle.

10.4.4 The Radix-8 Processor

A Conventional Radix-8 Processor

A good starting point would be to discuss the operation of a conventional arithmetic radix-8 processor. Such a processor would need to receive eight complex operands, multiply each operand by a twiddle factor, and perform an eight-point

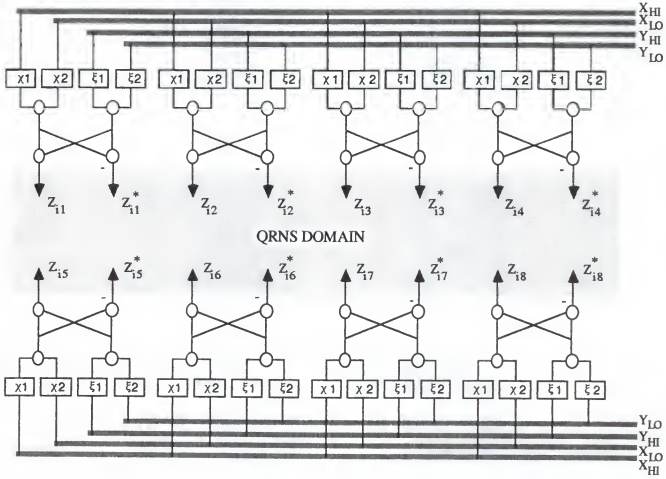


Figure 10.3. QRNS Forward Conversion Chip

DFT on the "twiddled" data. The twiddle multiplications are important because they allow the twiddled data to be passed to an eight-point DFT which is *identical* each time it is called.

Thus, a conventional radix-8 processor would consist of two basic parts: a multiplier array to perform the twiddle factor multiplications and an eight-point DFT engine. For a 512-point complex FFT, there are 64 eight-point DFTs for each of three stages, which means a total of 192 eight-point DFTs and 192 eight-point twiddle factor multiplications. Since the twiddle factors are the same for each 512-point FFT, they can be stored in a twiddle factor memory which is 192 words deep by 8 words wide. Also, the eight-point DFT can be implemented by a radix-2 decimation-in-time FFT which has three stages of four radix-2 butterfly computations. Since the eight-point radix-2 transform is identical, its twiddle factors can be stored in registers in the radix-8 engine.

The primitive computational element in the radix-8 engine is the radix-2 butterfly. The function performed by the radix-2 butterfly is structurally of the form

$$\begin{aligned} X &= x + wy \\ Y &= x - wy \end{aligned}$$

where x, y, are complex data and w is one of the complex radix-2 twiddle factors. Also, the eight-point FFT needs to be preceded by an array of eight radix-8 complex twiddle factor multipliers. A layout of a conventional radix-8 processor is shown in Fig. 10.4.

It is important to remember that a complex multiplication requires 4 real multiplies and two real adds. These operations must be performed by separate multipliers an accumulators if we wish to achieve as high a throughput as possible. In the computation of the radix-2 butterfly, the complex product wy can be formed and then

X and Y can be generated with two complex additions. Thus, the radix-2 butterfly element requires four multiplier units and six adders. Also, each twiddle factor multiplier in the array requires four multipliers and two adders. Since there are a total of twelve radix-2 engines and eight twiddle factor multipliers, we need a total of 80 multipliers and 88 adders.

Ignoring for now the required adders, we can assess the impact of the multipliers on the proposed design. It is clear that the multiply delay is the limiting factor in the pipeline rate. We can thus design with the fastest 16-bit multiplier available. This would appear to be the Cypress Semiconductor CY7516, which has a 38 ns clock cycle. Each of these parts consumes roughly 1 W so that we have dedicated roughly 80 W to the multipliers and limited the clock to a maximum of 38 ns. If we were to cascade three radix-8 processors (one processor dedicated to each of the three stages of eight-point FFTs), this would increase to 240 multipliers, consuming roughly 240 W.

10.4.5 QRNS Radix-8 Processor

The radix-8 FFT is computed as a nested sequence of eight-point FFTs. Twiddle factor multiplications are used to "tie" the small transforms together. It is important to remember that *every* eight-point transform is identical and that only the radix-8 twiddle factors change. Thus, a highly efficient radix-8 engine is crucial to the design of the FFTAP.

The radix-8 processors perform all of the FFT computations in the FFTAP. Recall that the QRNS decomposes computations into 8 parallel channels, each comprised of two subchannels. The computations performed in all of the channels are identical; only the residues are different. Furthermore, the computations performed in the

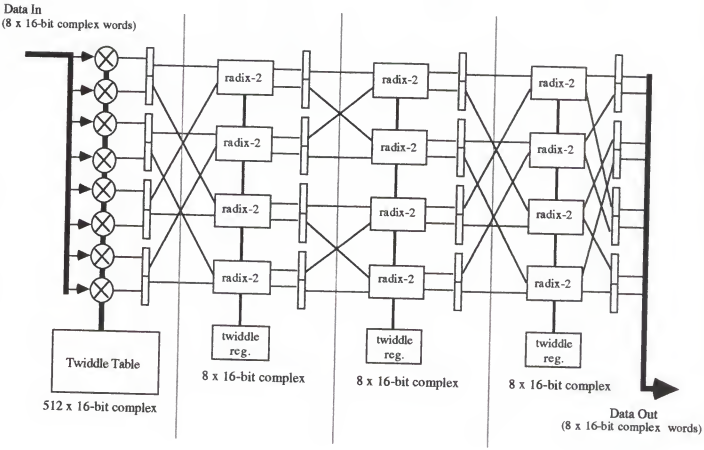


Figure 10.4. Conventional radix-8 processor

subchannels of the same channel are also identical. Thus, we will discuss the operation of only one radix-8 processor and applies equally to either the I or Q processor in any of the eight channels.

As with the conventional radix-8 processor, the QRNS radix-8 processor consists of two basic sections: the radix-8 twiddle factor multiplication array and eight-point FFT section. The radix-8 twiddle factor multiplications are performed by indexed addition and hence use QRNS multipliers (which are extremely small and have a clock rate limited only by a small table look-up cycle). The Input QRNS residues have been delivered by the input conversion subsystem. In order to avoid having to convert the complex radix-8 twiddle factors to QRNS format, they are stored in a small local memory in QRNS exponent format to be delivered eight at a time to the QRNS multiplier array. This memory is organized as a twiddle table which is 192 words deep by 8 words wide (64 words deep by 8 words wide for each of three FFT stages) where each word is only seven-bits. The total memory for the radix-8 twiddle table is thus 1536 seven-bit words.

The radix-8 processor is configured as an array of radix-2 butterfly units. The layout of the radix-8 processor physically mimics the organization of an eight-point decimation-in-time FFT, which is diagrammed in Fig. 10.5. The bit-reversal and data flow are hard-wired into the array.

Each radix-2 butterfly performs a computation of the form

$$\begin{aligned} X &= x + wy \\ Y &= x - wy \end{aligned}$$

The w are radix-2 twiddle factors and are the same for *every* eight-point transform in the entire FFT. Thus, the multiplication by w can be hard-wired into a table in each radix-2 engine. Hence, the butterfly computations can be performed with one

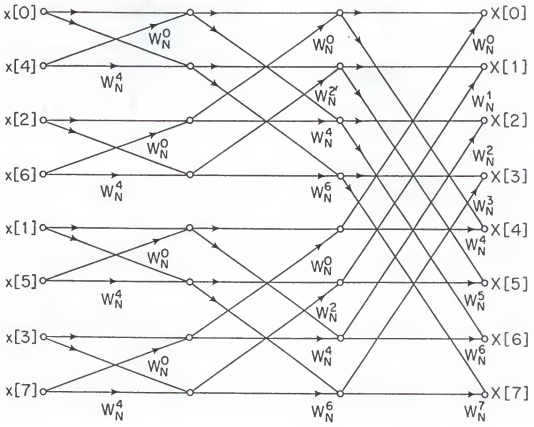


Figure 10.5. Eight-point radix-2 decimation-in-time FFT

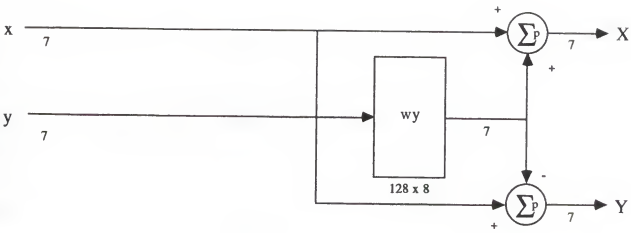


Figure 10.6. QRNS radix-2 butterfly

table to form the product wy mod p, a mod- p adder to compute $x + wy$ mod p, and a mod-p subtractor to compute $x - wy$ mod p.

The QRNS radix-2 butterfly is shown in Fig. 10.6 and is seen to consist of one 128×8 table and two modulo-p adders. The radix-2 butterfly is multiplierless and uses the table for radix-2 twiddle factor multiplication. The complete radix-8 engine consists of twelve radix-2 engines, for a total of twelve tables and 24 mod-p adders.

The radix-8 twiddle factor multiplication array consists of eight QRNS multipliers. If the twiddle factors are stored in exponent form, each QRNS multiplier consists of a table for QRNS-to-exponent conversion, a modular adder, and a table for exponent-to-QRNS conversion. Thus, the multiplier array consists of 16 tables and 8 modular adders.

In total, the QRNS radix-8 processor is comprised of 28 128-by-8 tables, 32 modulo-p adders, and 1536 words of memory for the radix-8 twiddle table. The QRNS radix-8 processor is shown in Fig. 10.7. The layout is structurally similar to

the conventional radix-8 processor. However, the data manipulated by the QRNS radix-8 processor is small in wordwidth and the hardware is multiplierless. A much higher throughput can be sustained by the QRNS radix-8 processor since its clock speed is limited only by the look-up time of a small table. Additionally, even though there are a total of 16 QRNS radix-8 processors in parallel (two per channel times eight channels), the size and power consumption are significantly less than that of the 80 multipliers needed to build a conventional processor.

Notice that the radix-8 processor is naturally pipelined and that a new set of eight operands can be delivered to its input each clock cycle. After the initial latency, a complete twiddled eight-point FFT appears at the output each clock cycle.

10.4.6 Scaled CRT subsystem

The scaled CRT subsystem is responsible for converting the QRNS outputs of the radix-8 processors back to complex integer format. Recall from Chapter 3 that a scaled CRT unit receives as its input all of the RNS residues corresponding to an integer and returns as output a scaled version of the integer.

The QRNS, however, requires that we first map the QRNS pairs to CRNS residues. Symbolically, we are applying the function ϕ_M^{-1}, which is a composition of the mappings $\phi_{p_i}^{-1}$, $i = 1, 2, \ldots, 8$. This produces the CRNS residues, the real and imaginary of which can then be passed to the scaled CRT to produce the real and imaginary scaled outputs. In our design, as demonstrated in Chapter 4, the mapping ϕ_M^{-1} and the scaled CRT are combined in the table contents to yield the function Π_M. Thus, a scaled QRNS CRT unit receives as input all of the QRNS pairs and produces scaled versions of the real and imaginary parts of the complex word.

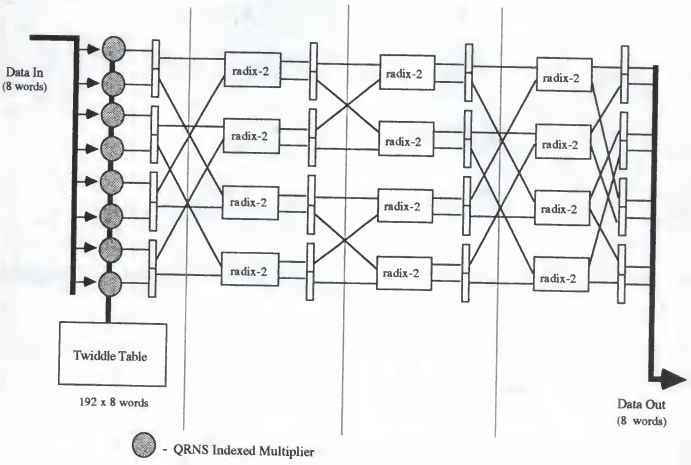


Figure 10.7. QRNS radix-8 processor

The FFTAP requires that eight complex words be recovered simultaneously. Our design is based on the use of eight scaled QRNS CRT units, each of which performs the output conversion for one of the eight complex words. Hence, there are eight scaled QRNS CRT chips, producing the functions $\Pi_M(l)$, $l = 1, 2, \ldots, 8$.

Notice that chip l receives all of the QRNS pairs corresponding to the l-th complex word. In other words, Π_M needs to communicate with both the I and Q channels. Since each output conversion chip can process a new set of QRNS pairs each clock cycle, the array of eight output conversion chips can accommodate the eight sets of QRNS pairs arriving each clock cycle once the initial latency of the radix-8 processors has transpired. Hence, after the initial latency of the output conversion chips, eight scaled complex outputs will be delivered per clock cycle.

10.4.7 FFTAP in the Cascade Mode

Recall from the first section that the FFTAP, when operating in standalone mode, processes 512-point FFTs in three stages. Each stage involves 64 eight-point procedures. The procedure involves a parallel multiplication by eight radix-8 twiddle factors followed by an eight-point FFT. Each FFTAP radix-8 processor contains a 192-by-8 word memory required for radix-8 twiddle factor multiplications: 64 words deep for each of the three stages by eight words wide. Hence, the FFTAP also has the ability to act as a "single stage" processor since it contains as a subset of its memory the contents necessary to perform on one stage if so desired.

In other words, three FFTAPs can be cascaded to triple the throughput. In standalone mode, the data needs to make three passes through the radix-8 processors, each pass corresponding to a single stage of the FFT. In cascade mode, the first FFTAP processes the first stage of the radix-8 FFT, the data is passed to the second

FFTAP which acts as a second-stage processor, and finally, the third FFTAP acts as the final stage in a radix-8 FFT. Note further that there is natural pipelining between the FFTAPs; after the first FFTAP has performed its 64 eight-point procedures and passed the data to the second FFTAP, it can accept a new time series. The same is true for the second and third FFTAPs. The throughput of such a configuration is one complete 512-point complex FFT every 64 clock cycles after the initial latency of the system has transpired. At a 20 ns system clock, this translates into 780K FFTs per second!

10.4.8 FFTAP Memory Subsystem

The data delivery and storage needs of the FFTAP are demanding enough to merit the design of a dedicated memory subsystem. Up to this point, we have not discussed how to deliver eight complex words to the input simultaneously or how to write the eight complex words emerging from the scaled CRT subsystem to the correct memory locations. At first, this may seem like an unmanageably difficult problem. The symmetry of the radix-8 FFT algorithm, however, can be exploited to yield a very elegant solution to the memory management problems associated with the design of the FFTAP.

Assume that the input sequence has been sorted into three-bit reversed order. That is, from the sequence $\{x(i)\}$, form the new sequence $\{X^{(0)}(i)\} := \{x(i')\}$ where the mapping from i to i' is as defined in Eq. 10.5. There are three stages to the radix-8 FFT algorithm, and the analysis of the memory management scheme is illustrated by the structure of the computations at each of the three stages.

Each stage consists of 64 calls to the radix-8 processor. During the first FFT stage, the processor needs to be sent eight data points of $X^{(0)}$ whose indices are

separated by one each time it is called. In other words, during the first stage, there are 64 calls to the processor, the first consisting of data points 0,1,2,3,4,5,6, and 7, the second consisting of points 8,9,10,11,12,13,14, and 15, and so on until the sixty-fourth call with points 504,505,506,507,508,509,510, and 511. After a group of eight points has been processed, the data are returned to their original locations. After the first FFT stage, the data has been transformed into some intermediate sequence $X^{(1)}$.

The second stage of the radix-8 FFT also consists of 64 calls to the radix-8 processor. Now, each time the processor is called, it needs to be sent eight data points of $X^{(1)}$ whose indices are separated by eight. Thus, during the second stage, there are 64 calls to the processor, the first consisting of data points 0,8,16,24,32,40,48, and 56, the second consisting of points 1,9,17,25,33,41,49, and 57, and so on until the sixty-fourth call with points 455,463,471,479,487,495,503, and 511. After a group of eight points has been processed, the data are again returned to their original locations. After the second FFT stage, the data has been transformed to a new intermediate sequence $X^{(2)}$.

The third, and final stage of the radix-8 FFT also consists of 64 calls to the radix-8 processor. Each time the processor is called, it needs to be sent eight data points of $X^{(2)}$ whose indices are separated by 64. The first call consists of data points 0,64,128,192,256,320,384, and 448, the second call consists of points 1,65,129,193,257,321,385, and 449, and so on until the sixty-fourth call with points 63,127,191,255,319,383,447, and 511. After a group of eight points has been processed, the data are again returned to their original locations. After the third FFT

stage, the data has been transformed to a new sequence $X^{(3)} = X$, which is the 512-point FFT of the sequence x. The points of X are in the proper order, as well.

The process can be visualized as follows:

$$x \xrightarrow{\iota'} X^{(0)} \xrightarrow{\text{stage I}} X^{(1)} \xrightarrow{\text{stage II}} X^{(2)} \xrightarrow{\text{stage III}} X^{(3)} = X.$$

We have designed a memory that is capable of meeting the needs of the FFTAP. The memory is organized as 512 eight-bit words in a cube. The memory is addressed in eight-word blocks available in x, y, or z directions, as shown in Fig. 10.8

If the cells are numbered as shown in Fig. 10.8, a block in the x direction will consist of data points whose indices are separated by one, a block in the y direction will consist of indices separated by eight, and a block in the z direction will consist of indices separated by 64.

Thus, if the sequence $X^{(0)}$ is stored in the cubic memory as shown, stage I fetches blocks in the x direction and returns them in the same way to produce $X^{(1)}$. Now, $X^{(1)}$ is sitting in the cubic memory so that stage II fetches blocks in the y direction and returns them in the same way to produce $X^{(2)}$. Finally, $X^{(2)}$ is sitting in the cubic memory so that stage III fetches blocks in the z direction and returns them in the same way to produce the 512-point FFT X.

The data is complex and 16 bits in width. In order to allow the memory to function at the system clock rate, the cubic memory module is partitioned into four cubic submodules: one for the high eight bits of the real data, one for the low eight bits of the real data, one for the high eight bits of the imaginary data, and one for the low eight bits of the imaginary data. The addressing and movement of the data is identical in all four submodules.

To allow one memory read and one memory write to occur within a single machine cycle, two such memory modules are used. In standalone mode, the sequence of reads and writes is as follows. During the stage I, data is read from the first memory and written to the second memory. During stage II, data is read from the second memory and written to the first. Finally, during stage III, data is read from the first memory and the 512 FFT points can either be written to the second memory or to some other output device.

In cascade mode, the memory function is similar. At the i-th stage, the processor reads from the first of its memories and writes to the first memory of the $(i+1)$-th processor. While this is happening, the processor at stage $(i-1)$ is writing to the second memory at stage i. After 64 clock cycles, the processor at the i-th stage reads from the second of its memories (which has just been written to by the processor at stage $(i-1)$) and writes to the second memory of the processor at stage $(i+1)$.

Symbolically, the procedure is simple to understand. There are six memories: 1_A and 1_B belong to FFTAP1, 2_A and 2_B belong to FFTAP2, 3_A and 3_B belong to FFTAP3. Let x and y correspond to two wavefronts of data (512-point time series). Recall that after initial latency, a 512-point record will pass from one FFTAP to the next every 64 clock cycles. The record x takes the path

$$x \longrightarrow 1_A \longrightarrow 2_B \longrightarrow 3_A \longrightarrow X$$

and record y takes the path

$$y \longrightarrow 1_B \longrightarrow 2_A \longrightarrow 3_B \longrightarrow Y$$

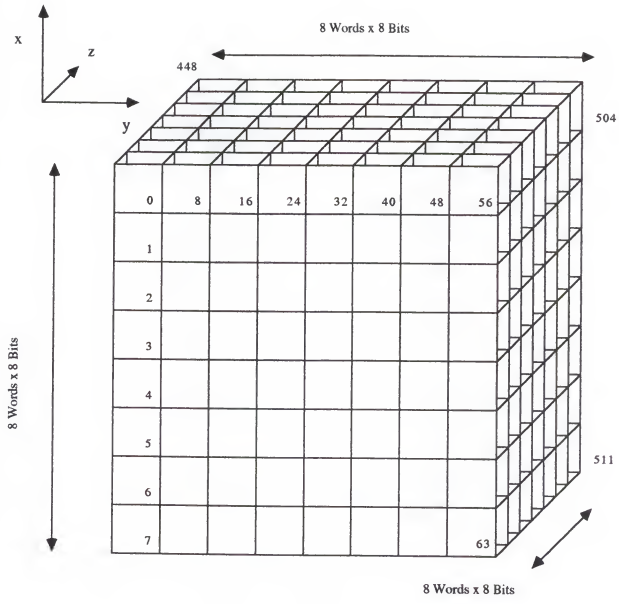


Figure 10.8. Cubic memory module for FFTAP

10.5 VLSI Layout and Timing Analysis

There are three basic chip types in the FFTAP: input conversion, radix-8 processor, and scaling CRT. All of the chips within a given subsystem are structurally identical; only the table contents are different. Hence, a study of the VLSI requirements of the FFTAP should include a detailed analysis of each of these chip types.

In order to achieve the packaging density necessary to partition the design of the FFTAP into the specified chips, it is assumed that the medium of choice is 1.0 micron CMOS. Our VLSI team has extensive experience with the design of 2.0 micron CMOS QRNS devices and can provide reasonable size and power estimates based on a scaling of present technology.

10.5.1 Input Conversion Chip

In Chapter 4, it was shown that the primitive QRNS input conversion element consisted of four 256×8 tables and four modulo-p adders (which translates to eight binary adders). Thus, a forward conversion chip which is capable of performing the forward mapping for eight moduli must consist of 64 eight-bit adders and 32 256×8 tables. Although this amounts to only 64 Kbits of memory, it is organized into small ROMs whose size tends to be dominated by the size of the address decode logic.

Fortunately, the data flow within the conversion elements is highly regular and there is no communication between conversion elements. This results in a minimum of interconnection within the chip which allows easy layout. The chip takes a 16-bit complex input (for a total of 32 bits) and delivers the seven-bit QRNS pairs for eight moduli (for a total of $14 \cdot 8 = 112$ bits). Thus, not including clock or power or ground, the forward conversion chip requires 144 pins. The size and input/output needs of this chip can be met by a 1 cm. × 1 cm. frame.

The input conversion chip is pipelined so that a new complex word can be presented to the chip each clock cycle. After the initial latency (which is equal to the latency of a single QRNS forward conversion element), a complete set of QRNS residues is delivered each clock cycle. Since the input conversion subsystem consists of eight input conversion chips in parallel, the timing of the subsystem is identical to that of a single chip. This is shown in Fig. 10.9.

10.5.2 QRNS Radix-8 Processor

In Chapter 6, it was shown that the radix-8 processor comprises twelve radix-2 elements, eight QRNS multipliers, and a 192×8 word table. The radix-2 element consists of one 128×8 table and four 8-bit binary adders. Each QRNS multiplier consists of two binary adders and two 128×8 tables. Hence, the radix-8 chip needs 28 128×8 tables, 64 8-bit adders, and the 192×8 word table.

The input to the radix-8 processor is one-half of the QRNS pair for eight words, which constitutes 56 bits. The output is the same. Not including power, ground, or clock, the radix-8 processor thus needs 112 pins. Again, it should be possible to meet the density and input/output needs with a 1 cm. × 1 cm. frame.

Recall that the radix-8 processor is pipelined so that a new eight-point time series is accepted each clock cycle, and after the initial latency, an eight-point FFT is delivered each clock cycle. The timing of the radix-2 and radix-8 engines is shown in Fig. 10.10

10.5.3 Scaled CRT Chip

In Chapter 4, it was shown that the scaled CRT chip for the output conversion of one complex word required sixteen 128×16 tables, twelve 16-bit adders, and sixteen

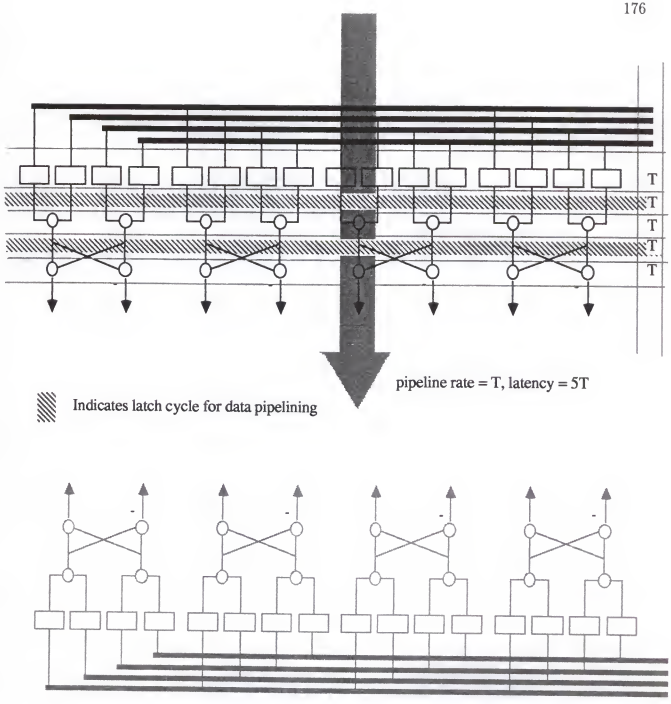


Figure 10.9. Timing analysis of QRNS input conversion chip

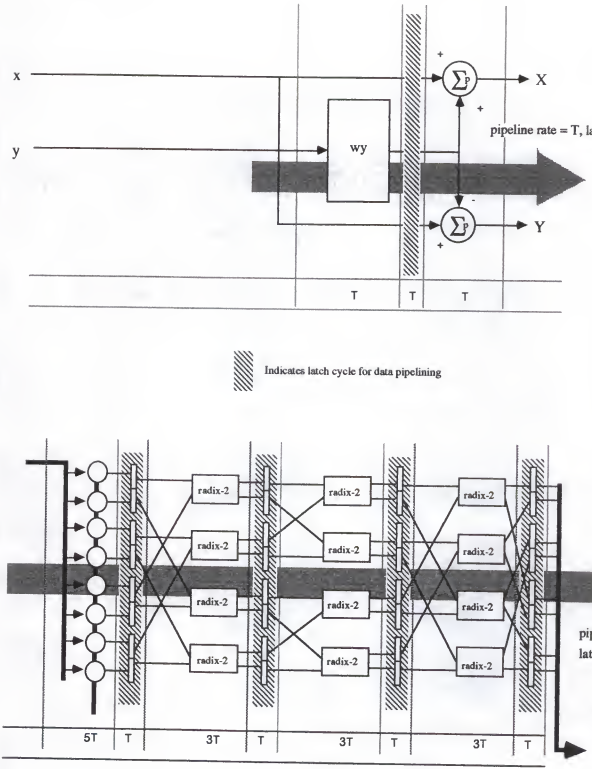


Figure 10.10. Radix-8 and radix-2 engine timing analysis

modulo-p adders. The 16-bit adders can be made from two 8-bit binary adders as can the modulo-p adders. Additionally, the 128×16 tables can be partitioned as two 128×8 tables. The scaled CRT chip then consists of 32 128×8 tables, and 56 8-bit adders.

The input to the scaled CRT chip is the complete set of QRNS pairs for one complex word, for a total of 112 bits. The output is the 16-bit real and 16-bit imaginary parts of the scaled complex word. Thus, not including clock, power, or ground, the scaled CRT chip needs 144 pins. As with the input conversion chip, the data flow is highly regular with a minimum of inter-cell communication. It should be possible to meet the density and input/output needs of the scaled CRT chip with the same 1 cm. × 1 cm. frame.

As with the other chips, the scaled CRT chip is pipelined to accept a complete set of residues each clock cycle, and after the initial latency, delivers a complex output each clock cycle. Since the scaled CRT subsystem consists of eight CRT chips in parallel, the timing is identical to that of a single chip. This is shown in Fig. 10.11.

The timing of the entire FFTAP for a *single* FFT stage is shown in Fig. 10.12.

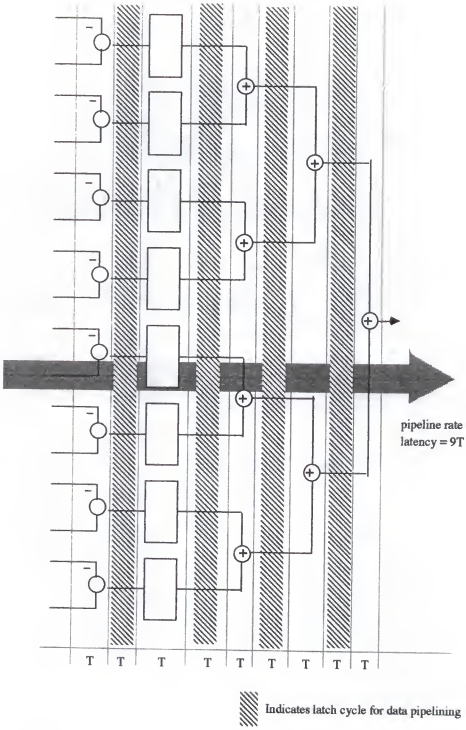


Figure 10.11. Timing analysis of scaled CRT chip

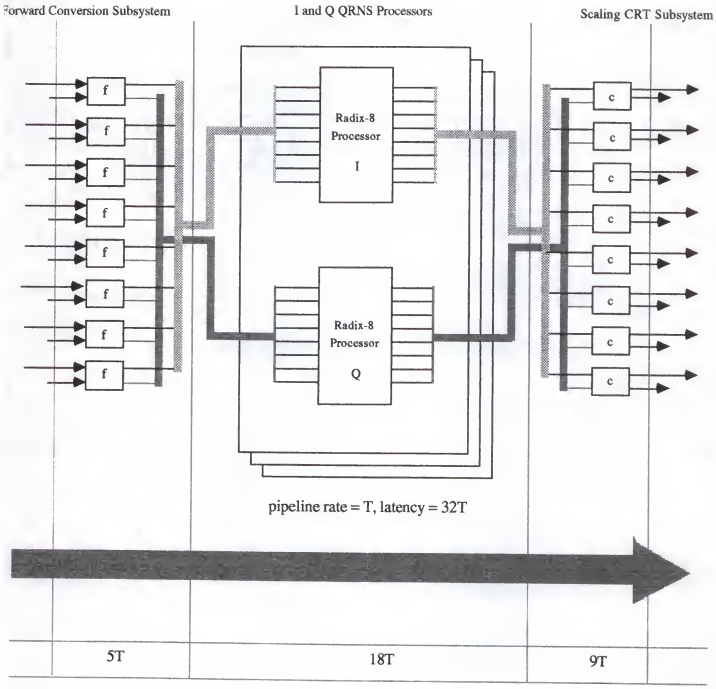


Figure 10.12. Timing analysis of FFTAP for a single FFT stage

The VLSI complexity and latency of each of the FFTAP chip types can be assessed by referring to the following tables. It is assumed that the clock period is given by T and that pipeline registers are used between tables and adders.

chip type	ROM size	# ROMs	adder size	# adders	i/o pins	latency
Input Conv.	256 × 8	32	8-bit	64	144	5T
Radix-8	128× 8	28	8-bit	64	112	18T
CRT	128×8	32	8-bit	56	144	9T

chip type	no. of chips
Input Conv.	8
Radix-8	16
CRT	8

Not including the cubic memory module, the FFTAP consists of a total of 32 chips. Assuming a low power dissipating process, a power dissipation of 10 W per chip is a very conservative estimate. Thus, the entire FFTAP should consume less than 320 W.

10.6 Numerical Simulation of the FFTAP

In order to simulate the input-output of the FFTAP, a fixed-point analysis of the radix-8 FFT needs to be performed. It is immaterial that the internal data representation of the FFTAP is the QRNS. The only relevant fact is that the numeric system is fixed-point in nature. What follows, then, is an assessment of the radix-8 FFT in a fixed-point environment on a suite of test signals.

The radix-8 FFT was simulated using SIGLAB, an interpretive DSP language from The Athena Group, Inc. The SIGLAB environment allows DSP routines to

be written in a powerful, high-level language, tested for fixed-point behavior, and graphically interpreted. The code for the radix-8 FFT is as follows.

```
function radix8(x,m,n)
wordlen(m,n,0) # wordlength= m-n integer bits, n frac. bits
m1=zeros(512)
m2=zeros(512)
xf=<zeros(512),zeros(512)>
radix8=<zeros(512),zeros(512)>
temp=<zeros(8),zeros(8)>
#
# create radix-8 twiddle factor masks
#
for(i=0:7)
  for(j=0:7)
    for(k=0:7)
      m1[64*i+8*j+k]=exp(<0,-2*pi*j*k/64>)
    end
  end
end
for(i=0:7)
  for(j=0:63)
    m2[64*i+j]=exp(<0,-2*pi*i*j/512>)
  end
end
#
# 3-bit reversal
#
for(i=0:7)
  for(j=0:7)
    for(k=0:7)
      xf[i+8*j+64*k]=x[64*i+8*j+k]
   end
  end
end
#
# first stage of 8-point DFTs
#
for(i=0:63)
```

```
  for(j=0:7)
    temp[j]=xf[8*i+j] # data indices are separated by 1
  end
  temp=gzdft(temp) # call fixed-point DFT
  for(j=0:7)
    xf[8*i+j]=temp[j]
  end
end
#
# multiply array by first stage of radix-8 twiddle factors
#
xf=fxpt(xf.*m1) # fixed-point multiply by twiddle mask 1
#
# second stage of 8-point DFTs
#
for(i=0:7)
  for(j=0:7)
    for(k=0:7)
      temp[k]=xf[64*i+j+8*k] # data indices are separated by 8
    end
    temp=gzdft(temp) # call fixed-point DFT
    for(k=0:7)
      xf[64*i+j+8*k]=temp[k]
    end
  end
end
#
# multiply array by second stage of radix-8 twiddle factors
#
xf=fxpt(xf.*m2) # fixed-point multiply by twiddle mask 2
#
# third stage of 8-point DFTs
#
for(i=0:63)
    for(j=0:7)
      temp[j]=xf[i+64*j] # data indices are separated by 64
    end
    temp=gzdft(temp) # call fixed-point DFT
    for(j=0:7)
```

```
      xf[i+64*j]=temp[j]
    end
end
radix8=xf # return value of function
end
```

The routine "gzdft" is a fixed-point DFT routine given by

```
function gzdft(x)
len=max(dim(x))
gzdft=<zeros(len),zeros(len)>
# create vector of frequencies around unit circle
w=exp(<zeros(len),-2*pi*rmp(len,1,len)/len>)
#
# Horner's rule for (fixed-point) polynomial evaluation
#
x=rev(x)
for(i=0:len-1)
  ww=w[i]
  sum=x[0]
  for(j=1:len-1)
   sum=fxpt(x[j]+ww*sum)
  end
  gzdft[i]=fxpt(sum)
end
end
```

The following test signals were generated and transformed using a fixed-point radix-8 FFT. The data format was a 24-bit word with 14 fractional bits and 10 bits for integer and sign. Two figures of merit were used for the error. First, since the errors were all approximately zero-mean, the variance of the error is used as a measure of the "error power". Second, the "relative error power" between the fixed- and floating-point spectra is defined as the ratio of the variance of the error to the zero-th lag of the autocorrelation of the floating-point spectrum.

1. 512-point sinusoid with a normalized frequency of 0.25.

2. 512-point sinusoid with a normalized frequency of 0.25 plus Gaussian noise with unit variance and zero mean. The signal-to-noise ratio is roughly 10 dB.

3. 512-point chirp with a normalized center frequency of 0.1.

4. 512-point chirp with a normalized center frequency of 0.1 plus Gaussian noise with unit variance and zero mean. The signal-to-noise ratio is roughly 15 dB.

5. 512-point sum of two sinusoids with normalized frequencies 0.125 and 0.25 plus Gaussian noise with unit variance and zero mean. The signal-to-noise ratio is roughly 10 dB.

The results are summarized in the following table.

signal	SNR	P_{err}	P_{rel}
sine	n/a	-67 dB	-91 dB
sine + noise	10 dB	-58 dB	-83 dB
chirp	n/a	-56 dB	-77 dB
chirp + noise	15 dB	-56 dB	-76 dB
two sines + noise	10 dB	-59 dB	-85 dB

It We conclude that the fixed-point FFT is quite effective in resolving spectral peaks and swept (fm) components even in the presence of noise. The test signals are characteristic of the type of signals that occur in signature and vibration analysis and indicate that the FFTAP could be quite effective in such applications.

CHAPTER 11
SUIMMARY AND CONCLUSIONS

The early sections of this dissertation presented the theory of computation by homomorphic images (CHI). The Chinese remainder theorem (CRT) and the theory of finite fields were the most important concepts we used in our schemes for CHI. It was shown that the CRT is an important tool which can be used to design extremely fast, small, and efficient computational units which are capable of meeting the demands of modern high-speed digital signal processing.

It was also shown that all of our hardware designs (. *e.g.*, RNS, QRNS, and PRNS input converters, scaling CRT engines, PRNS DFT units, FFTAP radix-8 processor, RNS and QRNS ALUs, etc.) can be built with just two standard logic parts: binary adders and small read only memories (ROMs). This comes at the expense of ignoring the rare cases where some of the salient computations are shift-realizable or simplified in some other manner. What is gained, however, is a unified approach to RNS hardware design.

This unified approach is desirable from the standpoint of VLSI hardware design. All of our machines are designed with just two parts which have been extensively studied and characterized. The data flow through these parts is highly regular and easily synchronized. Thus, our computing machines are extremely simple to design and test. Furthermore, our machines can benefit immediately from improvements in device technology. Our designs are highly modularized and partitioned in a manner which allows easy scaling with reductions in device sizes. As VLSI technology becomes smaller and faster, these RNS components will experience an attendant

decrease in size and increase in throughput without any modifications to the basic design.

For these reasons, it is evident that a collection of highly optimized adders and memory cells is crucial. Rather than take an approach where special modulus sets are chosen for reasons such as shift-realizability or error detection properties, we recommend that designs be based on a single set of optimized adders and memories. If this approach is taken, RNS designs will be viewed as less exotic, easy to design and test, cost-effective, and practical solutions to a breadth of signal processing problems.

As an illustration of our approach, we designed a high-speed processor for the computation of FFTs using only a pair of standard cells. The FFTAP was shown to be capable of producing 512-point complex or 1024-point real FFTs at a sustained rate of 260K transforms per second in standalone mode or 780K transforms per second in cascade mode. This is obtainable in a package consisting of 32 chips which should fit on a single board. Modifications, such as the reduced-hardware FFTAP, reduce the chip count to 18, with an attendant decrease in throughput and power dissipation. We saw, however, that the FFTAP could benefit greatly from the use of multi-chip modules.

The FFTAP is also fault tolerant, and although fixed-point in nature, can still operate with a high degree of numerical precision. This is because of the use of the QRNS as its internal numeric representation. The QRNS lends the design modularity, extremely high throughput, and an enormous advantage in multiply/accumulate efficiency over systems using conventional numeric representations.

We presented numerical simulations which verified the efficacy of fixed-point FFT implementations in situations likely to occur in signature analysis. Additionally, the VLSI floorplans and layouts provide a proof-of-concept that the density required by the three principal components of the FFTAP (namely, the input converter chip, the

radix-8 processor chip, and the scaled CRT chip) can be met easily by current VLSI technology.

REFERENCES

[1] R. C. Agarwal and C. S. Burrus. Fast digital convolution using Fermat transforms. In *Proceedings of the Southwest IEEE International Conference Rec.*, pages 538–543, Houston, TX, 1973.

[2] R. C. Agarwal and C. S. Burrus. Fast convolutions using Fermat number transforms with applications to digital filtering. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-22:87–97, 1974.

[3] R. C. Agarwal and C. S. Burrus. Number theoretic transforms to implement fast digital convolution. *Proceedings of the IEEE*, 63:550–560, 1975.

[4] R. C. Agarwal and J. W. Cooley. New algorithms for digital convolution. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-25:392–410, 1977.

[5] H. H. Aiken and W. Semon. Advanced digital computer logic. Technical report, WADC, Cambridge, Massachusetts, 1959.

[6] V. S. Alagar and A. K. Roy. A comparative study of algorithms for computing the Smith normal form of an integer matrix. *International Journal of Systems Science*, 15:727–744, 1984.

[7] D. K. Banerji and J. A. Brzozowski. On translation algorithms in residue number systems. *IEEE Transactions on Computers*, C-21:1281–1285, December 1972.

[8] A. Baraniecka and G. A. Jullien. On decoding techniques for residue number system realizations of digital signal processing hardware. *IEEE Transactions on Circuits and Systems*, CAS-25:935–936, November 1978.

[9] F. Barsi and P. Maestrini. Error corrrecting properties of redundant residue number systems. *IRE Transactions on Computers*, C-22:307–315, March 1973.

[10] R. A. Baugh and E. C. Day. Electronic sign evaluation for residue number systems. Technical report, RCA, Camden, New Jersey, 1961.

[11] A. Bequillard and S. D. O'Neil. Systolic RNS computation of the two-dimensional discrete cosine transform in a ring of algebraic integers. In *Proceedings of the 20th Annual Conference on Information Sciences and Systems*, Princeton, NJ, March 1986.

[12] E. R. Berlekamp. *Algebraic Coding Theory.* McGraw-Hill, New York, New York, 1968.

[13] Garret Birkhoff and T.C. Bartee. *Modern Applied Algebra.* McGraw-Hill, New York, New York, 1970.

[14] Richard E. Blahut. *Error Control Coding Theory and Applications.* Addison-Wesley, Reading, Massachusetts, 1983.

[15] Richard E. Blahut. *Fast Algorithms for Digital Signal Processing.* Addison-Wesley, Reading, Massachusetts, 1985.

[16] D. Bungard, S. Dharand, T. Hopkinson, and P. McDonough. Adaptive quadrature processing using a VLSI-based RNS systolic FIR filter. In *Government Microcircuits Applications Conference (GOMAC-86)*, San Diego, CA, November 1986.

[17] P. W. Cheny. A digital correlator based on the residue number system. *IRE trans. Electronic Computers*, EC-11:63–70, March 1961.

[18] J. W. Cooley and J. W. Tukey. An algorithm for the machine computation of complex Fourier series. *Mathematics of Computation*, 19:297–301, April 1965.

[19] R. Cosentino. Fault tolerance in a systolic residue arithmetic processor array. *IEEE Transactions on Computers*, C-37:886–890, July 1988.

[20] J.H. Cozzens and L.A. Finkelstein. Computing the discrete Fourier transform using residue number systems in a ring of algebraic integers. *IEEE Transactions on Information Theory*, IT-31:580–588, September 1985.

[21] R. A. Games. An algorithm for complex approximation in $z[e^{2\pi i/8}]$. *IEEE Transactions on Information Theory*, IT-32:603–607, September 1986.

[22] R. A. Games. Complex approximation using algebraic integers. *IEEE Transactions on Information Theory*, IT-31:565–579, April 1986.

[23] Richard A. Games, S.D. O'Neil, and J.J. Rushanan. Algebraic integer quantization and conversion. Technical report, The MITRE Corporation, Bedford, Massachusetts, 1987.

[24] H. Garner. The residue number system. *IRE trans. Electronic Computers*, EC-8:140–147, June 1959.

[25] C. F. Gauss. *Disquisitiones Arethmeticae*. Yale University Press, New Haven, Connecticut, 1966. English translation by A. A. Clarke.

[26] I. J. Good. The relationship between two fast Fourier transforms. *IEEE Transactions on Computers*, C-20:310–317, 1971.

[27] R. Gregory and E. Krishnamurthy. *Methods and Applications of Error-Free Computing*. Springer Verlag, New York, New York, 1984.

[28] R. T. Gregory and D. W. Matula. Base conversion in residue number systems. In *3rd Symposium on Computer Arithmetic*, pages 117–125, 1975.

[29] M.F. Griffin, F.J. Taylor, and M. Sousa. New scaling algorithms for the Chinese remainder theorem. In *22nd Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, 1988.

[30] E. Grosswald. *Topics from the Theory of Numbers*. Macmillan, New York, 1966.

[31] G.H. Hardy and E.M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, Oxford, U.K., 1938.

[32] J. Howell and R. Gregory. An algorithm for solving linear algebraic equations using residue arithmetic I. *BIT*, 9:200–224, 1969.

[33] Nathan A. Jacobson. *Basic Algebra I*. W.H. Freeman and Company, New York, 1985.

[34] W. K. Jenkins. A highly efficient residue-combinational architecture for digital filters. *Proceedings of the IEEE*, 66:700–702, June 1978.

[35] W. K. Jenkins. Complex residue number system arithmetic for high- speed signal processing. *Electronics Letters*, 16:660–661, August 1980.

[36] W.K. Jenkins. Techniques for residue-to-analog conversion for residue encoded digital filters. *IEEE Transactions on Circuits and Systems*, CAS-25:555–562, July 1978.

[37] W.K. Jenkins. Recent advances in residue number techniques for digital filtering. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-27:19–30, February 1979.

[38] W.K. Jenkins and B.J. Leon. The use of residue numbers systems in the design of finite impulse response digital filters. *IEEE Transactions on Circuits and Systems*, CAS-24:191–201, April 1977.

[39] B. Johnson, C. Nowacki, and J. Vaccaro. A systolic discrete Fourier transform using residue number systems over a ring of Gaussian integers. In *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Procesing*, Tokyo, Japan, April 1986.

[40] G. A. Jullien, P. D. Bird, J. T. Carr, M. Taheri, and W. C. Miller. An efficient bit-level systolic cell design for finite ring digital signal processing applications. *Journal of VLSI Signal Processing*, 1:189–207, August 1989.

[41] Donald E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, Reading, Massachusetts, 1969.

[42] J.V. Krogmeier and W.K. Jenkins. Error detection and correction in quadratic residue number systems. In *26-th Midwest Symposium on Circuits and Systems*, pages 408–411, Puebla, Mexico, 1983.

[43] S. Y. Kung. *VLSI Array Processors.* Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[44] Serge Lang. *Algebra.* Addison-Wesley, Reading, Massachusetts, second edition, 1984.

[45] J. L. Langston and K. Hinman. The application of finite fields and residue numbers to digital signal processing. In *Proceedings of the IEEE National Aerospace and Electronics Conference (NAECON)*, pages 41–48, May 1985.

[46] John D. Lipson. *Elements of Algebra and Algebraic Computing.* Addison-Wesley, Reading, Massachusetts, 1981.

[47] D. Mandelbaum. Error correction in residue arithmetic. *IEEE Transactions on Computers*, C-21:538–545, June 1972.

[48] S. Lawrence Marple. *Digital Spectral Analysis with Applications.* Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

[49] J. H. McClellan. Hardware realization of a Fermat number transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-24:216–225, 1976.

[50] J. H. McClellan and C. M. Rader. *Number Theory in Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

[51] H. J. Nussbaumer. Digital filtering using complex Mersenne transforms. *IBM Journal of Research and Development*, 20:498–504, 1976.

[52] K. H. O'Keefe. A note on fast base extension for residue number systems with three moduli. *IEEE Transactions on Computers*, C-24:1132–1133, November 1975.

[53] Alan V. Oppenheim and Ronald W. Schaefer. *Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[54] A. Peled and B. Liu. A new hardware realization of digital filters. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-24:456–462, December 1974.

[55] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William P. Vetterling. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, Cambridge, U.K., 1986.

[56] C. M. Rader. Discrete Fourier transforms when the number of data samples is prime. *Proceedings of the IEEE*, 56:1107–1108, 1968.

[57] C. M. Rader. Discrete convolutions via Mersenne transforms. *IEEE Transactions on Computers*, C-21:1269–1273, 1972.

[58] C. M. Rader and N. M. Brenner. A new principle for fast Fourier transformation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-24:264–265, 1976.

[59] I. S. Reed and T. K. Truong. The use of finite fields to compute convolutions. *IEEE Transactions on Information Theory*, IT-21:356–359, 1975.

[60] Alexander Skavantzos. *The Polynomial Residue Number System and its Applications.* PhD thesis, University of Florida, 1987.

[61] Alexander Skavantzos and Fred J. Taylor. On the polynomial residue number system. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, February 1991.

[62] D. L. Slotnick. Modular arithmetic computing techniques. Technical report, Westinghouse Electric Corp., Baltimore, Maryland, 1963.

[63] M. A. Soderstrand. A high-speed low-cost recursive digital filter using residue number arithmetic. *Proceedings of the IEEE*, 65:1065–1067, July 1977.

[64] M. A. Soderstrand and G. D. Poe. Applications of quadratic-like complex residue number system arithmetic to ultrasonics. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 28A.5.1–28A.5.4, 1982.

[65] M. A. Soderstrand and C. Vernia. A high-speed low-cost modulo p_i multiplier with RNS arithmetic applications. *Proceedings of the IEEE*, 68:529–532, April 1980.

[66] M. A. Soderstrand, C. Vernia, D. W. Paulson, and M. C. Vigil. Microprocessor controlled adaptive digital filters. In *IEEE International Symposium on Circuits and Systems Proceedings*, volume 1, pages 142–146, 1980.

[67] M.A. Soderstrand, W.K. Jenkins, G.A. Jullien, and F.J. Taylor. *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing.* IEEE Press, New York, 1986.

[68] M.A. Soderstrand, C. Vernia, and J. Chang. An improved residue number system digital-to-analog converter. *IEEE Transactions on Circuits and Systems*, CAS-30:903–907, December 1983.

[69] W. T. Stallings and T. L. Bouillion. Computation of pseudoinverse matrices using residue arithmetic. *SIAM Review*, 14:152–163, January 1972.

[70] A. Svoboda. Rational numerical system of residual classes. *Stroje Na Zpracovani Informaci*, 5:9–37, 1957.

[71] A. Svoboda. The numerical system of residual classes in mathematical machines. In *Proc. Congreso Internacional De Automatica*, pages 419–422, Madrid, Spain, June 1959.

[72] A. Svoboda and M. Valach. Operational circuits. *Stroje Na Zpracovani Informaci*, 3, 1955.

[73] N. S. Szabo and R. I. Tanaka. *Residue Arithmetic and its Applications to Computer Technology.* McGraw-Hill, New York, New York, 1967.

[74] M. Taheri, G. A. Jullien, and W. C. Miller. High speed signal processing using systolic arrays over finite rings. *IEEE Transactions on Selected Areas in Communication*, 6:504–512, 1988.

[75] R. I. Tanaka. Modular arithmetic techniques. Technical report, Lockheed Missiles and Space Company, California, 1962.

[76] F. J. Taylor and C. Huang. A comparison of DFT algorithms using a residue architecture. *International Journal on Computers and Electrical Engineering*, 8:161–173, September 1981.

[77] F. J. Taylor, G. S. Zelniker, J. C. Smith, and J. D. Mellott. The Gauss machine. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Toronto, Canada, 1991.

[78] Fred J. Taylor. Large moduli multipliers for signal processing. *IEEE Transactions on Circuits and Systems*, CAS-28:731–736, July 1981.

[79] Fred J. Taylor and C.H. Huang. An autoscale residue multiplier. *IEEE Transactions on Computers*, C-31:321–325, April 1982.

[80] B. D. Tseng, G. A. Jullien, and W. C. Miller. Implementation of FFT structures using the residue number system. *IEEE Transactions on Computers*, C-28:831–844, November 1979.

[81] R. W. Watson and C. W. Hastings. Self-checked computation using the residue number system. *Proceedings of the IEEE*, 54:1920–1931, December 1966.

[82] K. D. Weinmann, M. A. Soderstrand, and S. Shebani. Evaluation of new hardware for a high-speed, digital, adaptive filter using the residue number system. In *Proceedings of the 16th Asilomar Conference on Signals, Systems, and Computers*, pages 187–191, Pacific Grove, CA, November 1982.

[83] Bernard Widrow and Samuel D. Stearns. *Adaptive Signal Processing.* Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[84] N. Wigley, G. A. Jullien, and W. C. Miller. The modulus replication RNS (MR-RNS): A comparative study. In *24th Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, California, 1990.

[85] N. M. Wigley and G. A. Jullien. On modulus replication for residue arithmetic computation of complex inner products. *IEEE Transactions on Computers*, 39:1065–1076, August 1990.

[86] S. Winograd. Some bilinear forms whose multiplicative complexity depends on the field of constants. IBM Research Report RC5669, October 1975.

[87] S. Winograd. On computing the discrete Fourier transform. *Mathematics of Computation*, 32:175–199, 1978.

[88] S. S. Yau and Y. C. Liu. Error correction in redundant residue number systems. *IEEE Transactions on Computers*, C-22:5–11, January 1973.

[89] D. Young and R. Gregory. *A Survey of Numerical Mathematics, Vol II.* Addison-Wesley, Reading, Massachusetts, 1973.

[90] G. S. Zelniker and F. J. Taylor. The fft array processor. Technical report, Rome Air Development Center, Rome, New York, 1990.

[91] Glenn S. Zelniker and Fred J. Taylor. The Chinese remainder theorem, associative algebras, and multiplicative complexity. In *24th Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, California, 1990.

[92] Glenn S. Zelniker and Fred J. Taylor. On the reduction in multiplicative complexity achieved by the polynomial residue number system. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1990. (In review).

[93] Glenn S. Zelniker and Fred J. Taylor. Prime blocklength discrete Fourier transforms using the polynomial residue number system. In *24th Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, California, 1990.

[94] Glenn S. Zelniker and Fred J. Taylor. Small blocklength discrete Fourier transforms using polynomial residue arithmetic. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1990. (In review).

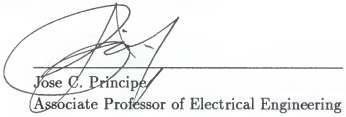
BIOGRAPHICAL SKETCH

Glenn Zelniker was born on December 1, 1965, in Boston, Massachusetts. He received the B.S.E.E. with high honors in 1987, the M.S. in electrical engineering in 1989, and the Ph.D. in Electrical Engineering in 1991, all from the University of Florida. Mr. Zelniker served as a research assistant at the Center for Mathematical System Theory at the University of Florida from 1987 to 1988 and as a research assistant at the High Speed Digital Architecture Laboratory at the University of Florida from 1988 until the present. He presently works as a research engineer at The Athena Group, Inc., in Gainesville, Florida.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

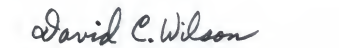
Fred J. Taylor, Chairman
Professor of Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.



Jose C. Principe
Associate Professor of Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.



David C. Wilson
Professor of Mathematics

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Kermit N. Sigmon
Professor of Mathematics

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Herman Lam
Associate Professor of Electrical Engineering

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

May 1991

for Winfred M. Phillips
Dean, College of Engineering

Madelyn M. Lockhart
Dean, Graduate School